A Graphical Environment for Gestural Computer-Aided Composition

by

EGON PASZTOR

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in Partial Fulfillment of the requirements for the degree of

Master of Science in Media Arts and Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

Author.....
Program in Media Arts and Sciences
September 3, 2002

Certified by.....
Tod Machover
Professor of Music and Media
Thesis Supervisor

Accepted by.....
Andrew Lippman
Chairman, Department Committee on Graduate Students

A Graphical Environment for Gestural Computer-Aided Composition

by

EGON PASZTOR

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in Partial Fulfillment of the requirements for the degree of
Master of Science in Media Arts and Science

Abstract

I have designed and implemented a software environment, a Windows application called Hyperscore, that presents a novel, easy-to-learn interface for composing richly textured music through line gestures. The program allows the user to command a set of music-manipulation algorithms written by Mary Farbood [Farbood, 2001].

The interface is both compelling and interesting for musically untrained users, and rich enough that such users, after sufficient practice, can create music that professional musicians find to be of high quality. While many musical composition programs geared for musically untrained users exist, it is its unique user interface, its use of freely drawn line-gestures, zooming navigation, and simple symbolic icons, that helps make this program unique.

The program was designed to enable musically untrained children, ages ten or older, to compose three-minute pieces for a string orchestra, given only a week or so of two-hour daily workshops. The program succeeded in this, and has been presented to audiences in Berlin, Dublin, and Glasgow as a part of Toy Symphony. The program has also been made available for download.

Thesis Supervisor: Tod Machover
Title: Professor of Music and Media

A Graphical Environment for Gestural Computer-Aided Composition

by

EGON PASZTOR

Thesis Readers:

Thesis Reader .....

Bruce Blumberg
Associate Professor of Media Arts and Sciences
MIT Media Laboratory

Thesis Reader .....

Joe Marks
Director of MERL Cambridge Research
Mitsubishi Electric Research Laboratory

Acknowledgements

I would like to thank my thesis advisor Tod Machover for his guidance and support during my time in his group, as well as my readers Dr. Bruce Blumberg and Dr. Joe Marks.

I'd like to thank Dr. William Freeman, formerly of MERL, now at the MIT AI Lab, who has had a strong influence on my academic life. Without him I would not be where I am today.

I would like to thank my mother who has supported me for many years.

I would like to thank my friends Diana Young, Ali Mazalek, and Zoe Teegarden, for their support and help while writing this thesis, and most of all I'd like to thank my girlfriend Maria Kamvysselis for her love and encouragement throughout this ordeal.

Table of Contents

"Most software systems are either easy to learn, or extremely powerful. Rarely are they both, for to be so demands that their rules of operation be simple, yet afford a boundless space of possible outcomes. This is difficult, and nearly contradictory. Nevertheless, there exist real-world exemplars of such systems, such as the piano and the pencil: although any four-year-old can discover their basic principles of operation, an adult can just as well spend fifty years practicing at them, and still feel like there remains more that can be expressed through them..."
-- Golan Levin [Levin, 2000]

Chapter 1: Introduction

With some art forms, the process of creating new works of art is viewed as an integral part of learning for the novice, even as he or she is just beginning work with the form. In the case of drawing, for example, the child is encouraged to produce her own original pictures as soon as she is able to pick up a crayon. Similarly, writing is taught concurrently with reading as a young student learns the literary arts.

Historically, music has differed from most of the other arts, in that composition is generally not considered a part of the curriculum for a child or novice. In many cases, the creation of music is viewed as a discipline to be taught and practiced only after the mastery of musical performance and theory. [Cope, 1996] One reason for this is that the articulation of music traditionally has involved complex mechanical instruments that require years of practice to gain expressive proficiency. Another cause is that writing musical scores involves abstract symbolic notation that is also difficult to learn. Thus, without considerable experience, a child who manages to conceive a tune "in his head" would have difficulty articulating that tune with his musical instrument and would have similar trouble recording it on paper. Based on the successes of past programs discussed in Chapter 2, we feel that technology can address both of these problems and enable the amateur to create content that she would not otherwise be able to create.

There have been many attempts to use software applications to reduce the complexity of musical composition. These attempts can be considered in one of two categories. First, some programs have provided painting or simple graphical tools to manipulate sounds, and have had great success as toys. However, the sound composed using such programs doesn't resemble traditional Western tonal music. [e.g., Metasynth] Other programs have focused on presenting the ideas of tonal music in a simple or "kid-friendly" way, but these programs tend to be either to simplistic to create complex

works or are too cumbersome to be used by a beginner. With Hyperscore, we seek a middle ground between ease of use and intricacy of result.

In software applications directed towards other fields, graphical sketching interfaces have been successfully utilized to lower the perceived complexity of difficult tasks. The program Teddy [Igarashi, 1999], for example, uses free-form line gestures to intuitively aid in the construction of 3D models, which must otherwise be built using complex CAD software. While Teddy's interface seemingly allows the user to create shapes using completely free movements, it is the way in which Teddy interprets the user's gestures that allows it to achieve simplicity of use: when a user sketches a line, the line creates a new 3D form or applies an extrusion or slice to an existing from, depending on the shape and manner of the gesture.

By adapting the sketching interface paradigm to make musical composition easier and simultaneously applying a rule structure based on music theory models, we seek to ensure that the users free-form input stays within the realm of the program's musical context. Hyperscore provides an environment in which novices can intuitively create musical compositions that sounds well-organized and above all, musical.

In this thesis, we present the design and evolution of Hyperscore, as well as discussions of its place in the field and suggestions for future work. Chapter 2 presents the background and motivation for Hyperscore and discusses various other related areas of research. Chapter 3 details the evolution of the program's design. Chapter 4 presents the features and operation of the program including details about the internal algorithms. Chapter 5 presents our main application and testing environment: the workshops of Toy Symphony. Chapter 6 presents an evaluation of the program and isolates areas that could benefit from future improvement.

Chapter 2: Background

This chapter provides an overview of other projects and programs that share key characteristics with Hyperscore, and provides some discussion about how these compare and contrast to the program at hand. This list of prior art is by no means complete, but provides a broad coverage of the many related fields.

As Hyperscore is a synthesis of several qualities, it is helpful to consider the intellectual predecessors of the several aspects of Hyperscore separately. Specifically, the program incorporates the following four ideas:

Hyperscore is software using

1. **line-drawing-graphics**, which enables
2. **composition-by-people-not-musically-trained**, making it also
3. **useful-for-teaching-traditional-music**, and uses a
4. **Zoomable-User-Interface**.

Below we discuss the significance of each of these aspects.

2.a Line-Drawing-Graphics

Computer interfaces that take freehand lines as input are generally called "sketching" interfaces. They are becoming popular because sketching input presents a number of important advantages over more constrained forms of computer interaction.

First, lines drawn with a pen, including a pen-tablet input device, have the potential to be very expressive, containing a great amount of subtlety and control while maintaining an organic, natural look. It can be hard to achieve the same effect with an editing system based around a mouse or the individual manipulation of control points. These can require a great deal of concentration or necessitate tugging at a curve in many different places to conform to a target shape. A pen can achieve a quality result much faster by utilizing the precision of the muscles in our fingers.

Second, sketching interfaces permit the user to trade quality for further speed, and produce rough outlines or informal first drafts of proposed content. Engineering drawings, product designs, user interfaces, 3D models,

diagrams, flowcharts and maps are all sketched out on whiteboards and pieces of paper by people as a normal part of communication and thought. Even though professional editors for many of these forms of content are available on computers, people tend not to turn to a computer until they've organized their ideas on pen and paper, because traditional interfaces for these professional editors do not facilitate the rapid informal specification possible with pen drawings.

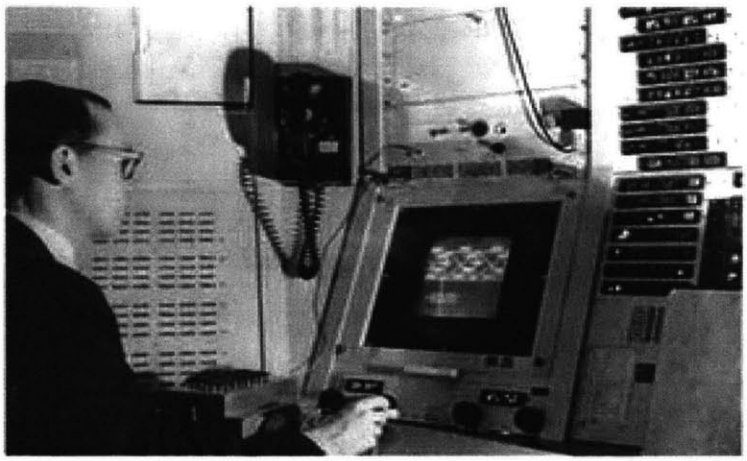


Figure 1: Ivan Sutherland at the console of the TX-2 - Sketchpad Project, MIT, 1963

The first sketching system was Ivan Sutherland's "Sketchpad" [Sutherland, 1963], designed at Lincoln Labs in 1963. In the days when computers were room-sized behemoths and CRTs were driven by oscilloscopes, "Sketchpad" permitted the user to draw simple shapes and lines freehand with a light pen on the screen. These freehand strokes were automatically "beautified": lines that were nearly straight, (according to some threshold), or nearly circular, or nearly parallel to other already-created elements, were automatically transformed to the ideal conditions they approximated. Far ahead of its time, the system also introduced simple pen gestures to trigger copying and pasting, and dragging an elastic rectangle to group objects.

Since "Sketchpad", other sketching interfaces have been explored for the creation of many forms of content, including 3D shapes, user interfaces, flowcharts, mechanical systems, and many others.

2.a.a Teddy and SKETCH

SKETCH [Zeleznik, 1996] and Teddy [Igarashi, 1999] are sketch-based 3D modeling systems that allow users to design 3D models through freeform drawing.

SKETCH, designed by Robert Zeleznik at the University of Blah in 1995, provides a gestural interface to polyhedral modeling, allowing the user to

construct, modify, and transform a 3D scene using gestural input. The system defines sets of specific pen gestures, taps and strokes, that are interpreted as commands to create various forms. (See figure 2) Further information is provided by where a gesture is made, what objects are nearby, and so forth.

In the second figure, (figure 3) a gesture is shown which inserts a new rectangle in a precise arrangement to a preexisting form.

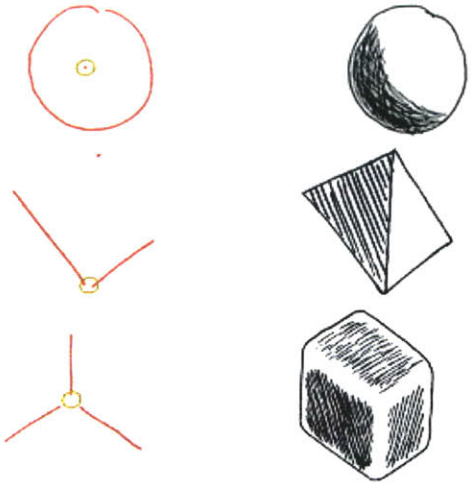


Figure 2. Within SKETCH, making the indicated gestures with the mouse causes the creation of the associated object

In order to use SKETCH effectively, the user must learn what gestures the system supports. However, because the gestural commands bear a similarity to the kinds of taps and motions a person would naturally make when drawing shapes with paper and pen, it is possible for novices to manipulate the program and perform complex operations that would require a much steeper learning curve on a more traditional CAD tool.

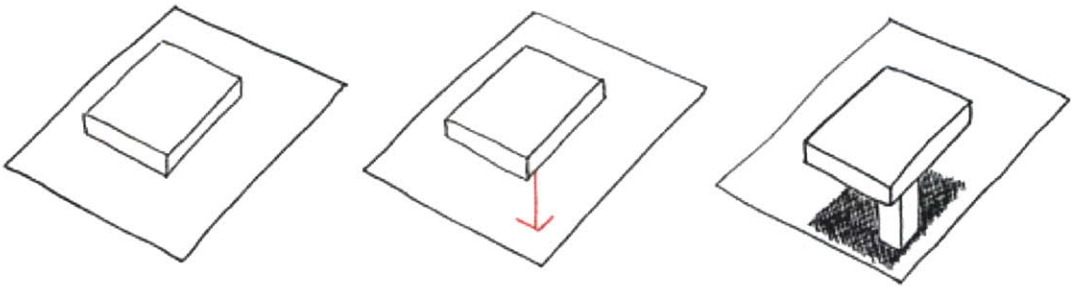


Figure 3. Initially, a tabletop lies on the floor. When the user draws a leg connected to a T junction, the program interprets this as a gesture to create a new primitive. The tabletop is automatically raised off the floor and the new leg touches both the floor and the underside of the table

While SKETCH focuses on polyhedral models featuring lines and planes, Teddy, designed by Takeo Igarashi and presented at SIGGRAPH 99, presents a gestural interface for constructing organic, rounded forms.

Within Teddy, a hand-drawn closed curve is automatically interpreted as the outline of a solid body, enabling the instant creation of lumpy round objects

or thin snakelike ones. Individual strokes can slice chunks out of the object, or cut holes and perform extrusions, with the result that a child using Teddy can easily create full 3D models of teddy bears, birds, plants and animals. Note that it is easier in Teddy to create organic curved forms than Cartesian

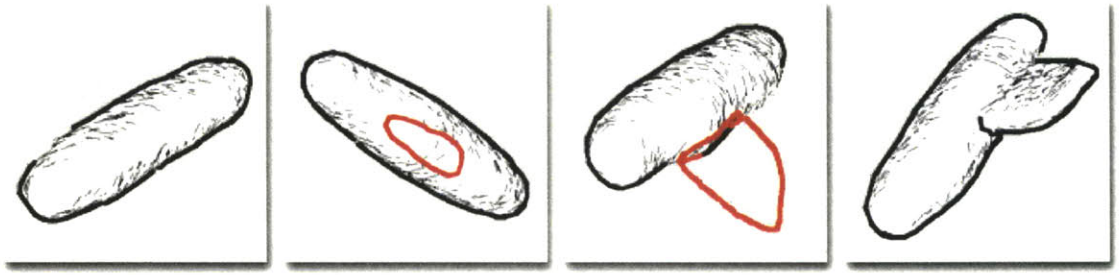


Figure 4. Two strokes in Teddy are interpreted as an extrude command. First the user draws the base of the extrusion. Second, the user draws its sweep. The system responds by adjusting the 3D model. The reader can try the program at http://www.mtl.t.u-tokyo.ac.jp/~takeo/teddy/teddy.htm

mechanical shapes like buildings or industrial parts, for these latter have straight lines and precise corners that are difficult to draw by hand. In contrast, traditional geometry-based modeling software tends to feature cubes and spheres as primitives, making mechanical parts easier to construct than organic forms.

Prior to Teddy and SKETCH, the methods of creating 3D models involved menu base Boolean solid geometry, the manual positioning of individual vertices in a polygon-mesh, or various sensor-based methods capable of scanning the geometry of a physical object. All of these tasks were so complex that only professional model designers typically created 3D models. These programs have helped make this form of content creation accessible to amateurs by allowing them to specify large amounts of shape information at once through drawing and gesture, eliminating the need to focus on individual polygonmesh points..

It is hoped that Hyperscore's simple drawing interface may have a similar effect upon the field of music composition.

2.a.b. Informal UI Design Tools

The informal, "rapid prototyping" capabilities of a sketching system are perhaps best illustrated by SILK , ("Sketching Interfaces Like Krazy"), a program which was the PhD work of UC Berkeley professor Dr. James Landay. Designed at CMU in 1995, the system presented an open canvas

upon which the user could draw boxes, lines, buttons, scrollbars and text, so as to aid in the initial construction of simple GUIs.

> "Sketching interfaces preserve the important properties of pencil and paper: a rough drawing can be produced quickly and the medium is flexible. However, unlike a paper sketch, the electronic sketch is interactive and can easily be annotated and modified using editing gestures.." -- James Landay, thesis abstract [Landay, 1996]

A close relative of SILK, Etchapad by Jon Meyer of NYU is an example of an informal interface-design program running on Pad++. The zooming canvas permits the user unlimited screen real estate along with the ability to add unlimited levels of detail.

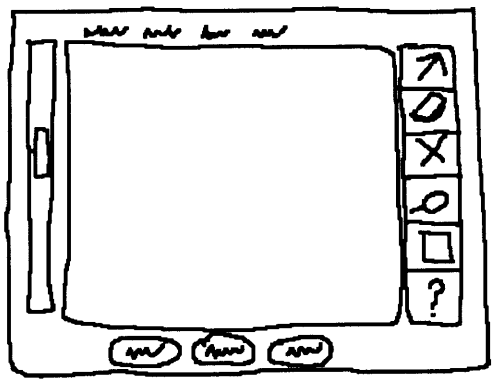


Figure 5. A sketched application interface created with SILK

2.a.c. Visual art by Golan Levin

Another use of sketching interfaces has been to focus on the artistic aspects of the strokes themselves. A series of visual art projects by Golan Levin [Levin, 2000] at the Media Lab in 2000 presented the user with a canvas upon which one could make marks. These marks would be subject to simulated physics laws in a number of different ways, reacting to the motion of the mouse or forces from nearby marks that were animating according to their own rules. The result of this was a visual experience, by which a user could create a moving kaleidoscope of color with a few strokes of his pen.

Others projects by Levin included audio components: the animated marks, in addition to vibrating and exhibiting other visual behaviors, would also cause auditory output. Levin's programs provided the user with an audiovisual artistic experience from just a few pen strokes, thus making it a philosophical cousin of Hyperscore.

Levin's systems used granular synthesis to produce sounds that directly related to the images. Hyperscore made use of physical strokes similar to Levin's in order to lend the canvas a responsive, realistic feel. However, Hyperscore seeks to use these to help kids compose music in the traditional tonal Western genre.

2.b Composing by the Musically Untrained

There have been several programs designed to enable people without a classical musical background to compose music.

Some programs [Xenakis (UPIC); Wenger (Metasynth)] have incorporated a graphical, painting approach provide a canvas to be interpreted as a time-versus-frequency grid. This enables the user to very easily make frequency and amplitude contours and allows for flexible sound placement. A diagonally slanted line is interpreted as a smooth pitch-bent rising tone; a vertical line represents a sudden smacking sound, of short duration but containing so many frequencies it does not sound pitched.

Other programs [Iwai, 1996 (Musical Insects); Rice, 1998 (Stretchable Music)] have objects with musical behaviors that exist and interact with each other upon a canvas. Rather than mapping time to an axis on the page, time passes as the system evolves in two dimensions, making sound as the program progresses.

2.b.a UPIC

The UPIC system by Iannis Xenakis [Xenakis (UPIC)] may have been the first such visual-music program, a computer system with a graphic input device that allowed composers to specify sounds and structures by drawing lines and shapes. UPIC has been used by numerous composers, including Jean-Claude Risset, Cort Lippe, Joji Yuasa and others, and a good deal of music produced by the system is available. In 1985, Les Ateliers UPIC, renamed CCMIX in 2000, was founded to promote the creative use of the system.

In addition to painting, a number of image-manipulation routines with awareness of the auditory properties of related frequencies could be used to apply a wide variety of "audio effects" to the visual image.

2.b.b MetaSynth

A recent implementation of the time-versus-frequency idea is MetaSynth by Eric Wenger; [Wenger; Metasynth], creator of the landscape design software Bryce.

Figure 6. A screenshot of MetaSynth. The image region in the center is treated as a time-versus-frequency plot of the output sound. Graphic editors allow the user to create complex textures and intricate rhythms..

MetaSynth presents a region into which the user can paint, and adds capabilities for sampling and wavetables, additive, subtractive, granular and frequency modulation synthesis. The mapping between sound and image is direct and transparent to the user, and thus it's very intuitive and easy to learn.

In essence, Metasynth and similar programs do allow an amateur to compose, and it is quite likely that children would enjoy producing art with this program. However, it is particularly difficult to use these kinds of interfaces to create Western tonal music, as there is no concept of *key* or *scale*, or even *note*. For Western tonal music, this interface is inappropriate.

Any content-creation program that accepts input of one modality, such as keypresses or mouse gestures, and produces output of a different modality like sound, must concern itself with the mapping between these two very different regimes. Metasynth maps each pixel to an oscillator at a particular frequency, to be activated at a particular point in playback. Hyperscore, as discussed in Chapter 4, has a mapping approach that is more complicated.

2.b.c SimTunes and Musical Insects

A completely different approach to making a toy-like musical interface appropriate for kids is the program SimTunes, a commercial version of Musical Insects by Toshio Iwai, at http://www.iamas.ac.jp/~iwai/.

This piece is a tool for visual music performance, and allows users to "paint" large swaths of colorful dots upon the program's canvas. Unlike Metasynth, which interprets the dot's visual shape and position literally as a set of frequencies to output, SimTunes treats the dot as an instruction to play a note whenever one of the autonomic "insects" passes over them.

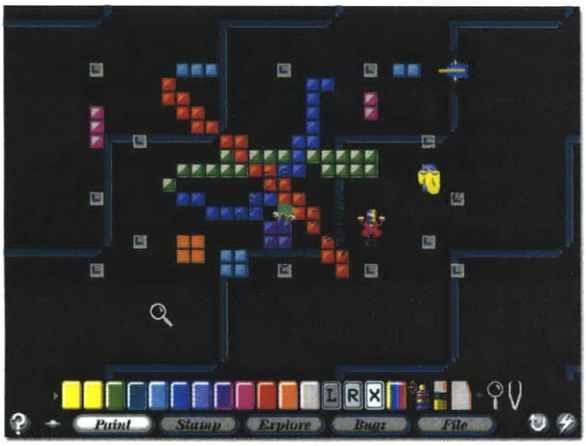


Figure 7. SimTunes. Each colored square produces a note when an insect walks over it. The user has trapped these insects in cyclic pathways.

The most unusual feature of SimTunes is that, instead of mapping time to a horizontal axis, the time at which a particular dot is played depends on when, or if, an insect gets to it. The insects control the sound output. Walking in one of four directions at a slow even pace, an insect plays any colored dots the user places in its path. Alternatively, the user can place baffles or barriers upon the canvas that cause the insects to turn; thus the user can trap an insect into wandering cyclic routes around the screen.

When the screen is filled with many colorful patterns, and multiple insects are walking around the screen in multiple paths of different lengths, the resulting rhythms can be quite complex.

> "These "music insects" as I call them, "react" to color dots on the screen. When they pass over such dots, they trigger musical scales, sounds, and different light patterns. The user selects colors from a palette with a track ball, and paints in their path, and the insects perform the colors when they pass over them. The insects' direction can be changed with certain colors, and colors can be painted to achieve less random musical "performances." -- Toshio Iwai [Iwai, 96]

2.b.d Stretchable Music

"Stretchable Music," [Rice, 98] created by Pete Rice at the MIT Media Lab, was a graphic-based interactive music installation that appeared more like a video game than a traditional musical instrument. "Players" used game controllers to manipulate graphical objects on the screen that represent different layers of a piece of music.

As shown in the figure, Stretchable Music presented a canvas showing a number of colorful, animated shapes. These shapes produced pre-composed music. Users could grab and stretch these objects, and each exhibited different musical behavior in accordance with the amount of stretch.

Stretchable Music, like SimTunes, was designed for musical novices and was primarily focused on having fun with sound. Hyperscore shares this feature -- It is possible in Hyperscore to produce large sections of pleasant musical texture with very little effort or concentration, and because of the ease of this process it might be considered entertainment. However, Hyperscore also aims to be useful as an educational tool. While children can just have fun making marks in Hyperscore, if they concentrate more carefully, they can learn many aspects of musical structure.

Figure 8: Stretchable Music. Each object on screen produced its own sound, and could be tugged and stimulated by the user.

2.c Programs useful for Teaching Traditional Music

While some programs are designed as toys to let kids enjoy making sound, other programs for children are written with the specific intent of teaching classical music notation and structure.

2.c.a. Jeanne Bamburger's "Impromptu" software

A music professor at MIT, Dr. Jeanne Bamberger has worked with children and music teachers in schools, and has written a Macintosh program called "Impromptu" that is distributed along with her text, "Developing Musical Intuitions" [Bamberger, 2000]. The program allows children to place notes on a staff to form short motives, and to save these as little graphical blocks.

By arranging these blocks in a row, children can create longer pieces and can experiment with the sounds of different patterns. For example, a user can create an A pattern and a B pattern, and then listen to ABAB or AABB. Bamberger intends her program to be used in a classroom setting, and places a great deal of emphasis on understanding scales, keys, and other concepts of traditional music.

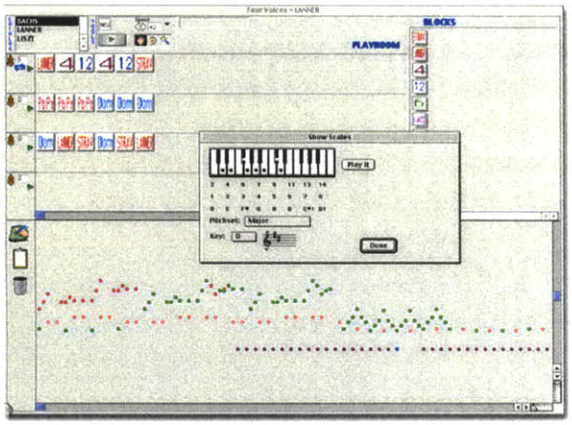


Figure 9: The user can define short motives, represented by the colored blocks, and arrange these blocks into patterns. The resulting piece is shown in paino-roll format at bottom.

Hyperscore shares many characteristics with Bamberger's system, including the ability to define short motives and arrange them into patterns at a higher level. Both programs possess a similar two-level hierarchy.

However, Hyperscore allows the user to very quickly alter a motive, by warping it to fit a particular contour or by applying new harmonization patterns. A large amount of novel material can be created from a single motive in a very short amount of time, because Hyperscore can treat each pen stroke as a transformation algorithm. Modifying a motive by a pen stroke is an "informal" process, in that the user does not explicitly decide on the new position of every note in the motive. By (optionally) taking these decisions away from the user, the user can give up some control over his composition in exchange for increased speed.

2.c.b. Morton Subotnick's "Making Music"

This program is flashier and more compelling than "Impromptu", but appears to lack the detailed pedagogical underpinning. "Making Music" [Subotnick, 1999] allows children to draw notes on the screen with a paintbrush cursor and appears very similar to the "motive windows" of Hyperscore. The application uses a familiar time-versus-frequency grid, with dots representing notes, and longer notes representing greater duration.

There are lots of programs with this type of interface, such as Sibelius and Finale [Websites referenced].

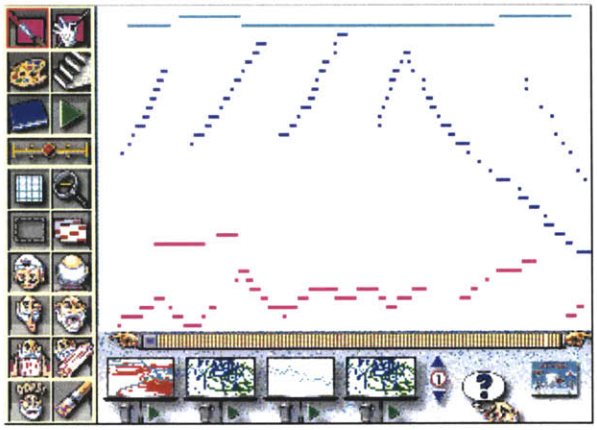


Figure 10: Colors represent notes on a time-versus-frequency grid. The user can lay down large numbers of these notes by dragging a pen.

These types of programs all tend to suffer from the same limitation, that it is difficult to compose significant amounts of interesting material without considerable training.

2.d Zoomable User Interface

The concept of the Zoomable User Interface (ZUI) was invented by Ken Perlin at NYU in 1995, where he first demonstrated this idea with Pad++ [Perlin, et al, 1996], a Tk/Tcl toolkit for making zoomable applications, including data viewers and web browsers.

The main advantage of ZUIs is their ability to represent large information spaces in a way that can be very quickly grasped by the viewer:

> As has been previously discussed, the main advantage of the zooming paradigm is that there is a strong sense of geography and object constancy. It becomes much easier to remember where you stored something rather than having to remember what you called your file or what folder you put it in. One remembers where in 'space' it is. This is because the human brain is much better at processing spatial rather than textual (or iconic) information. [Rashmi, 2001]

Due to the enormous complexity of a large composition, with multiple lines representing voices, lines delineating harmonization changes, and multiple motives, this approach to interface design seems fitting for Hyperscore.

The main disadvantage of ZUIs is that, if not designed carefully, the user can "get lost". If a user zooms in to view an object so close that the surrounding material, the context for the object in question, goes off screen, it becomes possible to forget where in the data-space the user is, and which way he's supposed to pan or zoom to find everything again. These problems must be considered when using this design paradigm.

Chapter 3: Design and Development

This chapter describes the process of designing and implementing Hyperscore, including an overview of the Toy Symphony project and Hyperscore's place within it, and the manner in which our goals and ideas changed as several early versions were developed before the final result.

Development on Hyperscore began in the winter of 2000, and continued until the Toy Symphony concert's opening night in February 2002, which provided a firm completion deadline. However, as with many software engineering projects, the precise specification for what the program was going to do and how it was going to do it was a moving target that evolved along with the developing program. We developed features in response to changing goals expectations as the research and the project progressed.

The purpose of this section is to describe the development schedule and the design considerations and choices throughout the work.

3.a Prehistory

Prior to mid-2000, when the graduate students of our group were just beginning to entertain an interest in musical devices for children, I was a student in a completely different group, Sandy Pentland's Vision and Modeling group, working on a vision system for a conference room outfitted with cameras and sensors. Therefore, I learned much of this intellectual prehistory secondhand.

Figure 11: The "Big Thing". A composition device that would look like this drawing was being designed prior to 2000.

However, even in those early days there was a prototype of the Shapers, the originals designed by Maggie Orth [Orth, 2001]. And there was a prototype of the Beatbugs [Weinberg, 1999], then called the Simple Things, little plastic egg-shaped things with buttons on them. To round out the set, a composition tool was desired.

The composition tool in existence then was called the Big Thing [Orth, 2001], a physical device that would allow kids to compose by plugging and unplugging large bendable tubes from colorful shapes A model had been constructed of the device as graduate students considered how it could be completed. After a great deal of thought, Tod Machover made the decision that the composition element of the Big Thing should be designed as software. Farbood began working on this, and I joined the group in December of '00, to work on the interface and graphical interpretation code.

3.b.a Initial Vision -- Klee's Lines

From the beginning, the expressive power of line strokes was the driving vision behind Hyperscore.

A line has many artistic properties to which we, as humans, are very sensitive. A line can be long or short, straight or curved. It can have sharp corners, or rounded edges; it can have lots of turns, or few. It can have small-scale textures; smooth, jagged, or ripply line. It can have overlayed waves on many scales. Using Wacom tablets for input, it is also possible to draw lines with variable thickness, thus adding yet another dimension for control. On the surface of things, it appears that with a single line an artist can specify continuous values for several separate functions along the line's length. In principle, each of these parameters could be mapped to a different musical parameter.

However, we were not interested in a direct mapping. The way we imagined the result in the beginning was that the (visual) *emotional effect* of a line could be made to match the (auditory) *emotional effect* of the produced music.

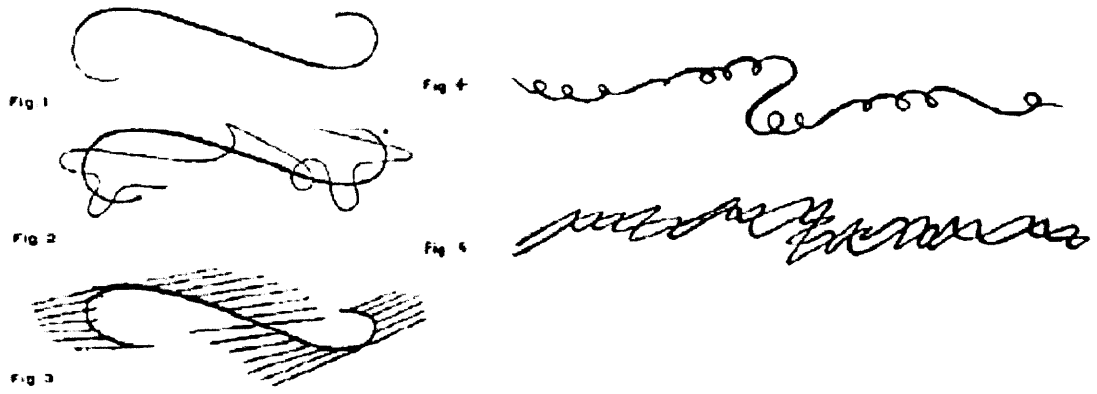


Figure 12: Examples from Paul Klee's "Pedagogical Sketchbook" of the expressive power of lines. "An active line on a walk, moving freely, without goal. A walk for a walk's sake."

The figure above vividly displays how a line can possess an emotional character. It was imagined that a user would draw such lines, whose emotional character varied throughout its arc. The system would interpret this affect and identify places where the affect changes. The music generation system would then produce music whose emotional qualities varied in the same way. The result would be a system where a pen-drawn arc would be interpreted as music.

3.b.b Toy Symphony

A plan was made for the development of our project within the schedule of Toy Symphony, a series of public events in the spring of 2002. The main event would be a family concert that would feature pieces showcasing our research. Pieces would be composed for the Beatbugs [Weinberg, 1999], the Shapers [Orth, 2001], and the Hyperviolin [Young, 2001], and amateur children would perform onstage in collaboration with a professional orchestra, demonstrating that innovative technology could make such collaborations possible. In addition to the concert, there would also be an Open House, where concertgoers could touch and explore our instruments for themselves, and a series of workshops where local children from schools would be taught to use our instruments in focused sessions. [Machover, 2001, Toy Symphony Summary Document]

The role of Hyperscore in Toy Symphony was to allow musically untrained children to compose, during the planned week of workshops, specifically *three-minute pieces for string orchestra*. We would have many children each working on their own pieces, and at the end of the workshop the best piece would be translated into traditional notation and played live in concert, completing the demonstration that, with the help of technology, children can compose successful music.

There were many unknowns, such as how many children we would have, how old they would be, and how much assistance they would need from teachers during the workshop week, but two things we had decided upon: The children would have no musical experience, and they would have one week.

That in the end Toy Symphony actually happened more or less the way it was planned in early 2000 is a tremendous achievement, and preparation for the concerts was a driving motivation in our program's design. It is a unique feature of the MIT Media Lab that many projects are designed for presentation to the public or to industry, in addition to (and sometimes instead of) publications and journals.

It was important that our program be a real advancement over existing composition-for-the-amateur tools, but it was also important that we be able to demonstrate this to regular concertgoers. It can be argued that designing a program for presentation outside the research environment demands a high level of robustness, for it must actually work in the real world with real people and not just within laboratory experimental setups.

Some scientists produce work so they can publish papers and become respected in the conference circuits. Other scientists produce work so they can impress the venture capitalists and boom startups. However, we had the public, critics, professional musicians, and perhaps most importantly, the kids to think about.

3.c The "*Green Line*" Version

We developed Hyperscore through three preliminary versions before the final version was completed and tested in the Toy Symphony concerts. The figure here displays the program that became our first version. It presented a window where the user could draw a single green curve -- smooth, or bumpy, with angles or waves. As the user drew or redrew this curve, boundaries and red marks would appear to indicate the stylistic boundaries between different regions of the line.

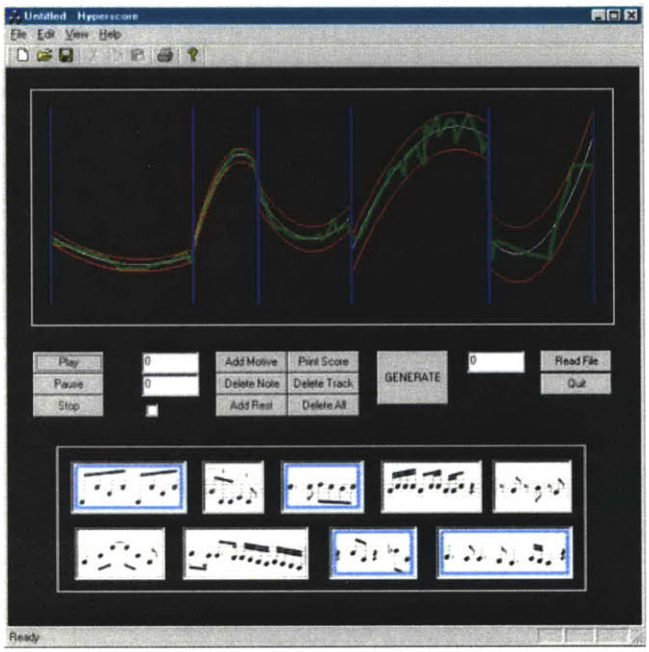


Figure 13: The "Green-Line" version. The user selected a motive, and drew a contour curve.

In essence, the program tried hard to fit the curve with a hierarchy of parabolas, and reported the best-fit breakpoints for a heuristic compromise between goodness of fit and minimal number of parabolas.

The user could select one or more motives from a set of nine that Farbood had composed and hard-coded in, and this motive would be used in generating the music. Then the user drew a stroke and hit "play" to hear the result. Farbood's music-generation system produced a kind of smooth tonal music that always sounded good, and tended to have more non-tonic chords in

sections where the green line had more shakiness. Farbood describes her algorithms for generating chord sequences in [Farbood, 2001].

However, while the music sounded good, the user had almost no sense of control. There *was* a perceptible difference between the music corresponding to smooth and bumpy sections of the line, but the user had to listen carefully to perceive it. Thus the user was not left with the sense that he had "composed" the music in any way.

3.c.a Literal Control versus Automatic Generation

Clearly, future versions of the program had to offer the user more control, but how much control was appropriate?

At one extreme, a program generates nice-sounding acceptable music all by itself, and the user's inputs cause the music to change in various ways. This has the advantage that kids could sit down at it and stroke right and left and draw any old thing, and hit play, and some pretty nice music would come out. However, it begs the question as to whether the user has actually done any composition; certainly, it is hard to feel a sense of accomplishment for music that arrives without any effort. And if the user thought that he wanted to make a change in a particular segment, to add or alter a particular sound, there was no way to make such a specific, directed change. This lack of control could breed frustration, dissatisfaction, and boredom.

We decided that we needed a more direct mapping between the marks left by the user and the output sounds. However, at this other extreme, if a program were to simply play everything that the user drew exactly as he drew it, regardless of whether or not it sounded good or made musical sense, then the risk was that making good music would be too hard. An interface that drew some staff-paper and permitted the user to place notes on it, (and there are many such programs on the market) is very easy to understand and provides a good sense of control, but does nothing to simplify composition.

The various prior-art for composition programs all seemed to fall to one of these two extremes.

Of the note-based programs, Simtunes [Iwai] and Impromptu [Bamberger], although very different in presentation, both give the user literal control over each note that is played, making it very difficult for a user to create complex, intricate textures. The Brain Opera applet [Machover, 96], on the other

hand, generated intricate music automatically but gave the user only a few parameters to slide back and forth.

Of the audio-based programs, Metasynth and UPIC, and Levin's work, literally transcribes some user's mark into the set of frequencies to output. The user has, by definition, complete control; but it's very difficult to make organized, successful music instead of frenetic theremin-like sounds. Stretchable Music, on the other hand, always sounds musical, but the user is limited to the few preprogrammed objects, and once these are familiar the novelty wears quickly.

Moreover, neither extreme would succeed as a part of the Toy Symphony concert. If the program did not provide enough help to ease the difficulty of composing good music, there was a risk that the Toy Symphony children would produce stuff that sounded *bad*, or if good, be very short. At the other extreme, where the program generates music mostly automatically, treating the user's marks as mere parameters, there tended to be too little variety.

3.d The "*Hasco*" Version

The first version had allowed the user to affect the chord progression through the line he drew, but only provided a single set of radiobuttons to choose the rhythmic material used in generating the music. A simple manner in which the user could be granted more control was to allow pieces that shifted between multiple motives during their run. This was the main functional improvement present in the second version.

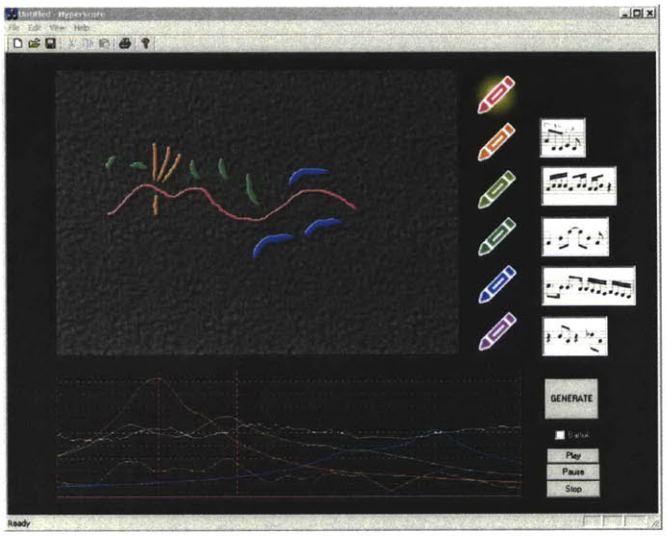


Figure 14: The "Hasco" version. The user drew a spine with a red pen. The other colors were associated with precomposed motives. Their proximity to the spine determined which motive was used to generate music.

In addition, the second version introduced a freeform graphical surface upon which to draw strokes, as well as the ability to draw multiple strokes, not just a single main line but "annotations", attendant strokes that would accompany the main line to give more information about the music there. However, while the user is able to specify a rich variety of information to the

program through multiple freeform strokes, only a few very specific aspects of the graphics actually affected the sound at this early stage.

The central spine acted just as it had in the "Green Line" version to determine the chord progression, except that it was no longer constrained to be a single-valued function with a horizontal ordinate. Rather the user was allowed to draw freely anywhere on the page, even producing loops or scribbles. However, internally the program measured the curvature of the central line as a function of its arc length from the user's starting point, and this curvature function was treated just as the user's function had been in the previous version.

Thus, if a user of the first version wanted to indicate that he wanted a great deal of chordal variation in a part of his piece, he would draw that part of his function such that it oscillated up and down rapidly as it progressed horizontally. In this second version, he would draw a section of his spine curve with rapid changes in curvature, wiggling from side to side. This is just as easy to do, but appeared to the user to be less constraining.

The annotations in this version performed only a single function, which was that the program determined, for any given point along the spine's length, which color annotation was closest. This selected a single motive, from six pre-composed motives, to apply at the corresponding point in the piece. However, while this feature didn't take advantage of much of the considerable information in a freely drawn stroke -- the system was aware of only the distance between the annotation and the spine -- it was planned that future versions would.

3.d.a Form before Function

At this point in the development of Hyperscore we were faced with very basic questions about the program's design. The user would draw a picture. That much had been fixed, and this picture would be abstract, containing lines and swirls in the manner of the Paul Klee drawings. But then, this picture would be interpreted as music. How could we translate abstract graphical symbols into sound without taking the literal time-vs-frequency-map approach of MetaSynth?

The "Hasco" version introduced the innovation that children were permitted to make any kind of stroke, even one that looped backward on itself or formed shapes. This opened the door to the user drawing symbols, dots or letters. Were we inventing a new notation with its own icons that the user would have to learn? There was a time when we considered that the system would

recognize different symbols, and do different things for a spiral than for a triangle.

In the end we decided not to interpret these strokes as symbols, because to do so would not play to the strength of strokes as an expressive tool. A stroke is a continuous function, or multiple continuous functions depending on how it's analyzed, and the richness of a sketch-based interface comes from the speed and accuracy with which the human hand can present these functions to the computer. The idea of recognizing specific shapes in the user's strokes strayed too close to the idea of "selecting one of a number of options", which could better be done with checkboxes, or a keyboard.

Figure 15: Should Hyperscore recognize these shapes as explicit symbols, and respond to them as a new form of notation?

Thus we decided to limit ourselves to recognizing continuously varying qualities of each curve, such as curvature or slope or height, and mapping them to parameters in the music. But the question remained as to what these parameters were going to mean.

3.d.b The Central Spine line

This question was most prevalent regarding the central line that served as the program's timeline. From the beginning Hyperscore had been developed with a sense that no matter how many lines the user drew, there would be one special single line that would "tell the story" of his piece, to which all the other marks would be annotations. We alternately called this the "timeline", later the "spine", later the "harmony line", but it remained fixed in all the versions as a feature we had decided we wanted to be central, though we varied, during the development, about what it was going to actually do, musically speaking.

We understood that a user with a pen could draw a line with both a dramatic overall sweep and small scale texture varying continuously. That is, the user could draw a line rippled or smooth or jagged, and vary this texture during his stroke while controlling the overall sweep of the stroke independently. In this way, the user could control two continuous parameters with a single line. Additionally, pen-based tablet entry would allow the user to vary pressure, or line thickness, a third parameter.

However, deciding what *musical* parameters to map to these *visual* parameters was difficult because our goal was that the music should possess an emotional feel similar to the emotional *look* of the line. Mapping the small scale texture to the degree of chord variation felt vaguely right -- frequent chord changes give music the sense that it is "going somewhere", or coming back from where it has recently gone, and doing this frequently can be described as the auditory effect of a line shaking back and forth. Mapping line thickness to overall volume (something we did in later versions) also makes a sort of intuitive sense; a thin line is "less there" in much the same way as a quiet melody is "less there".

But what should be mapped to the overall sweep? Imagine a spine that starts, vertical moving upward, and then makes a right turn in the middle? Or a spine that's just a straight line with three kinks in it? Or a long spiral with a loop in it? We could have mapped either coordinate, slope or curvature of the overall sweep to any musical parameter we could think of, but what would make the music *sound* like what the line *looked* like? These were frustrating issues.

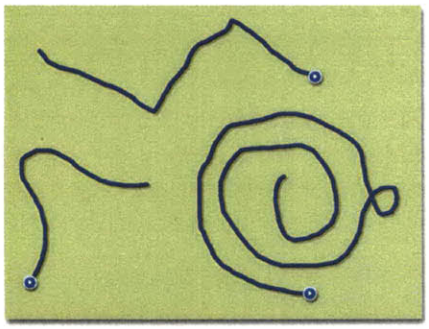


Figure 16: Potential spine curves. What kind of music sounds like each of these curves, respectively?

Eventually, the Evaluation Section of this thesis will argue that we ended up with a satisfying blend of all our desired features; a satisfying mix of computer assistance and user control, a satisfying mix between flexibility and ease of use. We ended up with a program that's uniquely good for composing an endless variety of tonal music without formal training. But perhaps the story of how we got there is at least as interesting as what we finally ended up with, if only so that others who travel our path will be the wiser. This stage in our journey was one of uncertainty and self-doubt, where we wondered whether our overall goal -- matching the music's emotional tone to that of the drawing -- even made sense.

3.d.b.1 Motive material constrains an emotional target

One reason mapping the drawing to an emotional tone might not have been possible, at least with the way our program was developing, was that the central line, which was theoretically supposed to determine the emotional weight of the piece, had no effect on the motive material out of which the piece was constructed.

In the Hasco version, there were only six precomposed motives, but they were placed into the piece by the presence of "annotative" strokes whose shape in that version was ignored. It was a simple matter to extrapolate to the future where the shape of the "annotative" strokes would affect each motive, and the motives would not be precomposed. Perhaps each "annotative" stroke could determine a motive in its entirety? Or perhaps they would only affect motives that the user would enter some other way.

But the fact remained that the user's ability to specify motives and arrange them in time fought against our desire to create a mapping between the graphical shape and the emotional feel. The nature of the motive material and its placement in time affects emotional feel, nearly completely determines it, even. If the spine's shape could only affect the chord progression chosen, or the harmony or the key, it would still not be enough to control how the music "felt" if the user chose motives incompatible with the desired emotion.

(Imagine the user designing motives with lots of fast notes jumping wildly in pitch, and placing these motives in large clumps along a spine drawn in the shape corresponding to "make me a sad sounding song". If the computer had to play the motives where they were drawn, the emotional target might be impossible.)

Alternatively, if we allowed the computer to alter the motives and their placement in any way necessary to achieve the target emotional feel, then the user would be left with the same lack-of-control problem mentioned above. How could the user specify a motive, if the computer was allowed to ignore it?

The user has to be able to reliably instruct the computer to play a group of notes at a certain time, and this requirement severely restricts the computer's ability to target a certain mood.

3.d.b.2 Most curves have uncertain emotional interpretation

The previous sections discussed the problems a computer might have creating music that seems "sad" or "happy", but in fact the real problem is more difficult than this due to the great variety and depth of human emotion and the daunting task of mapping these emotions to different kinds strokes.

Emotion in music is ideally able to convey a fantastically richer set of emotions; joy, satisfaction, smugness, jealousy, envy, anger, pride, love or hate and dozens more in great variety.

Broad heuristics would easily enable one to write code to distinguish a happy line from a sad one; similarly it is likely that some heuristics could be designed to give any music some positive or negative affect (overcoming the difficulty described in 3.d.b.1). But in order to claim the program did anything of note, it would have to have much more richness than merely recognizing positive from negative.

3.d.b.3 Difficulty in using stylistic property to position sharp boundaries

A final problem was that we persisted in thinking of section boundaries as sharp transitions. The "Green Line" version had separated the spine into sections with different styles and displayed their boundaries with vertical red lines. The "Hasco" version no longer displayed them, with the result that people no longer felt compelled to obsessively fiddle with their line in order to put the boundary in a specific spot, but the same algorithm was used internally.

The problem is that where one section in a line ends and another begins can be a highly subjective judgment. For example, consider a line which is straight, running horizontally halfway across the page, where it develops some ripples. Consider that this rippled/not-rippled transition may be used to delineate a change in the chord sequences to be played.

Figure 17: If the computer must determine a precise location for the transition from "flat" to "wavy", its decision will be highly subjective, based on the particular line-interpretation algorithm used. It would be hard for the user to manipulate this line in order to force the derived transition boundary to a single specific desired position (without destroying the smooth character of the line by introducing, say, an obvious corner).

You might imagine that the exact position where the ripples began would be something of subjective decision. It would help if the first ripple began at a sharp corner, but if the ripples started up smoothly, then it would be a judgment call.

It would become annoying if the user wanted to precisely set the position of this interpretive boundary, because the user would basically have to redraw the line, possibly over and over again, hoping that the computer would choose the transition boundary that the user wanted. An interesting approach would have been to explore fuzzy transitions where the exact position of the center was never displayed to the user. However, Farbood's musical

algorithms demanded sharp boundaries, and we wanted to allow the user to see where they had been placed.

However, this has the consequence that the user is frustrated trying to reshape his curve over and over again to nudge a transition based on a subjective interpretation toward a target value. If the user's goal is to position a transition boundary to a specific place, it should have been possible for the user to simply grasp the boundary itself and drag it there, rather than making its position a consequence of an interpretation of the spine.

3.d.b.4 The end of emotional interpretation

Throughout the early months of 2001, while we played with the "Hasco" version, the reaction from people to whom we showed it was mixed. On the one hand, people were intrigued by the free-form graphical appearance and saw that it had a great deal of potential, but they were frustrated that they couldn't really control the sound that came out. Despite being able to choose different motives at different times, and effect the chord progression with the spine, all the music "Hasco" produced sounded very similar, and our advisor kept telling us to increase the variety of the output music.

Through this feedback, and the difficulties we thought about, outlined above, the "Hasco" version basically marked the end of our trying to interpret drawn shapes for their emotive or musical content. From this point forward, we focused on treating lines as simple functions that were to be read literally, with a much lower degree of computer participation. The computer could "clean up" the notes that the user placed, but the user was ultimately responsible for each note that was played.

3.d.c Code Separation

The "Hasco" version also marked the point at which Mary and I decided on a strict separation between our two parts of the program.

While we talked about the functionality of the program together and with our friends and labmates, we were ultimately on two very different tracks. She was a composer interested in algorithms for automatic harmonization, thus her interest in generating different chord progressions. I am interested in interface design, and I was fascinated by the usability issues present in a large graphical content-creation program. While the specific problems of music were interesting, I was hoping to take out of the experience ideas that

would be of relevance to user interface design in general. Freehand sketch-based interfaces have great potential, and I wanted one that wasn't tied to a particular application, at least not too tightly.

Thus Farbood designed and implemented code for manipulating the complete musical score in memory, morphing motives and harmonizing whole swaths of notes, and I designed and implemented the GUI, the interaction, the look and feel and the framework. Together we wrote a single C++ header file to define a pair of pure virtual classes through which our code would communicate and which we updated as our design evolved.

It is this separation that permits me to write a thesis about the Hyperscore interface separate from Farbood's work, and this separation that allowed Farbood, last year, to publish her harmonization algorithm in her thesis independently from myself. Neither of us touched each other's work, or even knew much about its internals. Throughout our development it remained possible for me to compile and run the Hyperscore code without Farbood's work included. (Although the resulting program would make no sound.)

Thus, in this thesis you will find discussions about our various interface strategies, and, in Chapter 4, descriptions of various stroke-manipulation and interpretation algorithms used in the GUI. However, details about the specific harmonization algorithms used, of chords and scales and keys and such musical stuff, can be found in Farbood's thesis [Farbood, 2001].

3.e The "*Hyco*" Version

The third version was completed in the summer of 2001, and represented a dramatic shift toward more control for the user. While the spine and annotations were still freely drawn strokes that were not constrained to flow from left to right, in practice we began using them as such. While the spine's small-scale texture continued to influence chord progressions as before, the vertical position of the spine was mapped to volume, making it easy for the user to create pieces that got louder or softer as the piece played.

However, the greatest change to this version was the literalness with which the annotations added motives into the piece. In the "Hasco" and previous versions, at any given time the music had only one motive "associated" with it. Farbood's algorithm had taken this motive and generated multitextured, polyphonic harmonized music based on that motive, with the spine shape used in selecting the chord sequences.

All the annotations that the user drew (in the previous versions) simply determined, for any given point along the spine, which color motive was closest. For example, if the user had drawn a long yellow annotation parallel to the spine, then "yellow" would be selected for the whole piece. All other annotations, further away on the page, would have had no effect whatsoever.

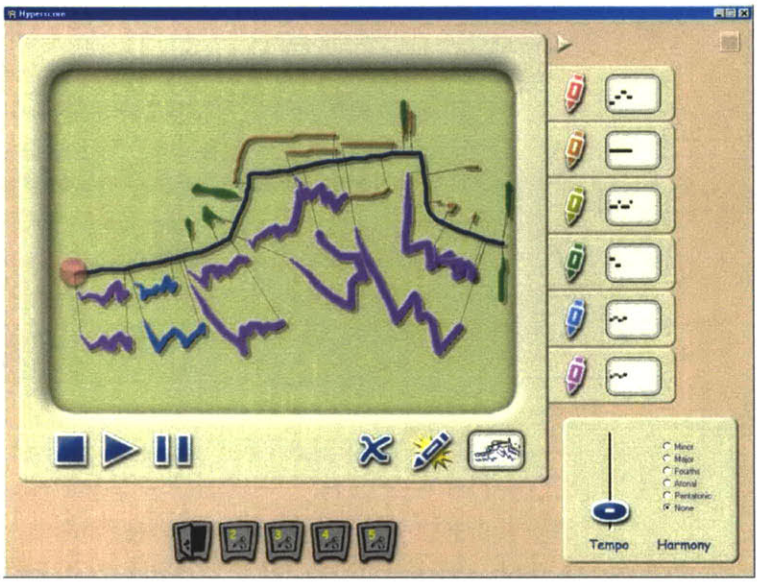


Figure 18: Each annotation added a motive into the final piece, just like the current version does. Little lines went from the edges of each motive to the spine. The spine's vertical position controlled overall volume.

In this version, however, gone was this automatic generation. Rather, every annotation had a simple literal effect of adding its associated motive into the piece at the precise time and pitch the user drew it. Each annotation was connected to the spine through a line with attachment handles that could be slid along the spine, thus the user could specify where on the spine a given annotation's effect was to begin and end. Each annotation caused its associated motive, appropriately altered by the shape of the annotation stroke, to be added to the resulting score, starting at the time corresponding to the position of the starting handle, and looping until the time corresponding to the position of the ending handle.

All the annotations added their motives into the score one by one in this manner to form a complete, unharmonized score, which was then harmonized according to the spine texture, if harmonization was turned on.

This design finally gave the user the sense of control the earlier versions had been missing. The program no longer generated music "all by itself". Rather, the user was responsible for placing each motive into the final piece. If a segment of the spine was left without motives, that section of the piece would be silent.

The shape of each annotation finally had an effect as well; each annotation was interpreted as a simple function, height as function of horizontal position, which was used to directly alter the contour of the motive. Thus, if a user drew a straight, horizontal stroke, the motive would play as designed. If

a user drew a stroke that angled upwards, the motive's latter notes would be pushed higher in pitch, altering its contour. In addition, an annotation's absolute vertical position set the absolute base pitch of the motive.

By using the annotations in this version, a user could transpose any motive up or down in pitch, alter its contour, have it loop for a specific period of time, and schedule it to start at any given time. The computer assists the user in two ways:

First, the manner in which each motive was reshaped by its annotation's contour required some careful coding, because we didn't want it to be possible for a user to completely destroy the motive's natural contour, no matter what shape the annotation was.

And second, the harmonization, which was a final pass that, in essence, worked over all the notes after they had been scheduled and positioned by the annotation curves, in order to fit them to a target key and chord progression.

3.e.a User-designed motives

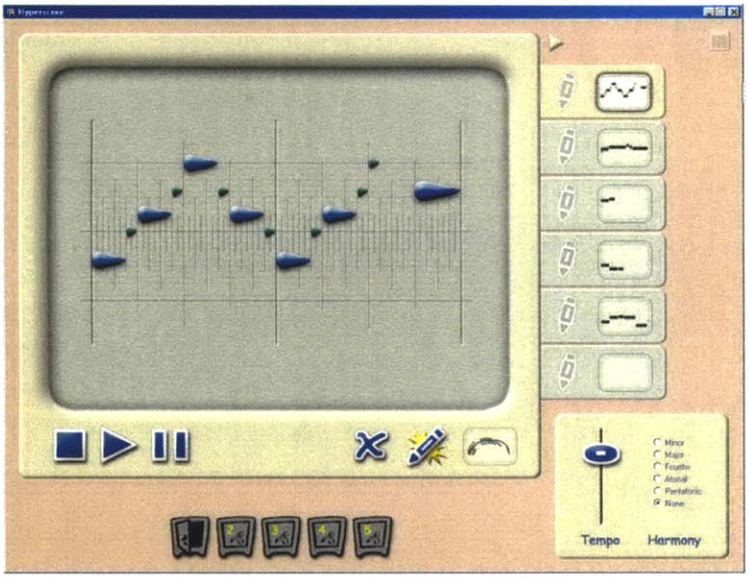


Figure 19: The user could edit each motive by moving little note-shaped graphics around

The "Hyco" version was also the first to offer the user the ability to create his own motivic material, which contributed greatly to the variety of music possible. When the user double-clicked on the motive thumbnail next to each pen, the view flipped to a motive editor.

The motive editor, displayed here, showed a fixed time-versus-frequency grid upon which the user could place little droplet-shaped figures of various sizes to represent notes. I chose not to use traditional staff notation because to do so would have meant presenting the user with a large number of iconic symbols whose meaning has to be learned. Traditional notation represents notes of various lengths with different symbols, open or filled-in circles, with and without

stems and flags, and so forth. It's an alphabet, which excels at compressing musical information into a form that can be read in real time by a player trained in sight reading. But for a novice at first exposure, it's a confusing mess.

Instead, I chose to represent notes of different duration directly by symbols whose lengths corresponded to the direction on the horizontal time axis because this mapping seemed very obvious to a novice like myself, and it has seemed clear to everyone I've shown it to.

In traditional notation, groups of notes are clustered and bound by visual elements like flags that connect related notes; and groups of short notes to be played together are expanded to take up more than their fair share of a measure in order to make them easier to read. The traditional notation appears optimized for rapid reading by experienced professionals, and my simplistic notation lacks these adaptations, but they are not needed in a tool where the user will only ever look at small static patterns.

The same can be said about the vertical axis. In traditional notation, the five lines represent the pitches of a key, and a composer can place notes in the non-key pitches by placing appropriate # or *b* signs. Again, this is optimized for speed reading -- five is probably the maximum number of lines that the human visual system can handle for instantaneously perceiving which line a note is near.

However, remembering which notes are in which key is something which may be nontrivial for someone musically untrained, and would be irrelevant here anyway: The precise vertical position of each note placed into the motive editor doesn't really matter very much, because the harmonization algorithm will alter the precise pitches when the motive is drawn into a piece.

This simple motive editor turned out to be a very successful addition to Hyperscore, and many of the children who later used it were observed to focus much more attention and concentration upon the motive views than the main view.

3.e.b Kevin Jennings

The lack of children testing the program at this point was a concern. It was August of 2001 by this point, and as our development had progressed, so too had the plans for Toy Symphony moved ahead. The first concert was to be in Berlin, in February 2002, and the orchestra which would play the Hyperscore pieces had been contacted and arrangements were being finalized.

With the February deadline nearing we needed to start planning how the workshop would go. How did one run a workshop? None of us knew anything about kids; we were programmers and musicians, not teachers. We had been saying that Hyco was a program that would be easy for a child to use, yet none of us had ever sat a child down in front of it and checked this out.

To this end Tod Machover asked Kevin Jennings, a PhD student at Media Lab Europe whom he had been advising for over a year in music pedagogy tools, to join Toy Symphony. Kevin is a music teacher, adept at working with children as well as composing, and was an invaluable resource. We would not have been able to pull off the Toy Symphony workshops without him.

Kevin visited the MIT Media Lab, saw the program, and was present for two of the three test workshops that were eventually run. Over the last months of 2001 while Farbood and I completed the software, Kevin wrote up teacher's guides explaining how Hyperscore could be used in a classroom setting. His guides explained what sequences of actions, followed by careful listening and discussion, would enable the teacher to present specific musical concepts to their students. In the end, Kevin was present at all our Toy Symphony workshops and taught many of the children himself, and helped to make our program accessible to the teachers. Hervé Gomez, a researcher at Media Lab Europe, also helped extensively with these workshops.

3.e.c Test Workshops

With Kevin's assistance we set up three short testing sessions during September and October with various children we could find, including Tod Machover's own children, Hana 7 and Noa 4, several of Hana's friends from school (7 and 8), and the two children (7 and 10) of Bahktiar Mikhak, another Media Lab professor who was interested in Hyperscore.

All three of these testing sessions basically went by the same way: we set up several computers in a spare conference room, set up a second table of milk and cookies, and waited for the children to come at the time we had arranged with their parents. We showed them the program, showed them the things it could do, and asked them to make pieces that they liked. In the middle we took a snack break; but we kept the children for about an hour and a half, after which we discussed what we'd seen.

Overall, we learned the following things from our test workshops:

3.e.c.1 The teacher is very important.

This was perhaps the most important lesson we learned, that no matter what form the interface took, good instructions and teacher participation could make or break the success of any workshop. A child who was kept focused by a teacher could likely produce great work with any interface, while a child left to play freely might goof off no mater how strong the interface. Thus Kevin's talents as an educator were very important.

3.e.c.2 10-year-olds focused on tasks much better than 7-year-olds.

Certainly there is some minimum level of mental maturity that is necessary to perform a composition task. Even if the interface is completely transparent the user needs to be able to keep a compositional goal in mind, to care about the result, and to focus his attention on the program long enough to try something and then see if it worked.

In our meetings and plans for Hyperscore up to this point we had spoken of 7-year-olds when we talked about the program's target age group. However, the 7-year-olds who visited us, while they were interested and cooperative and enjoyed the program, were so enthralled by the ability to make *any* mark on the screen or to make *any* sound, that asking them to carefully aim for a *particular* sound seemed to be a difficult goal. On the other hand, a 10-year-old boy who came to a test workshop was dramatically different, focusing intently on the motive and stroke views, listening to each mark as he drew it. The 7-year-olds were delighted by all the sounds, musical or not; while the 10-year-old was working toward a specific goal.

Hence we recommended that the children found for the Toy Symphony workshops average 10 years old, rather than 7.

3.e.c.3 More time was better.

We were, as mentioned above, intending to give the children a week of workshops to complete the target three minute string orchestra piece; but our back-of-the-envelope estimates, based on the speed of work we saw at our test workshops, left us discouraged. We began having meetings where we considered letting the children stay to use the program overtime, outside the workshops, at home or at school. They could take their pieces with them on floppy disks and work constantly.

After the Berlin concert, we became more confident, and in fact in Glasgow, in June of 2002, some fine pieces were composed in only four workshops days with certainly no out-of-workshop work. But this was not known at this stage.

We recommended that the individuals setting up the logistics in Berlin try to get the children some extra time on Hyperscore, before our workshop week was supposed to start.

3.e.c.4 The Motive-Editor focused concentration most of all

This was something we had noticed with our exceptionally focused 10-year-old boy; that even he treated the main stroke view as a painting tool where the sound of the strokes didn't matter tremendously. However, the motive editor focused his attention on the sound of the motive completely.

We were not sure whether this was because the motive editor offered less freedom, or because there was a grid in the background, or because it was shaded gray rather than exciting green. However, we felt it was probably a bit of all of these things, and we would benefit if our future versions aimed to make the stroke view closer in feel to the motive view, rather than, say, vice versa.

This was one of the main reasons that we made the decision to separate the flow of time from the spine curve in the final versions, and instead to make time flow left-to-right in both stroke *and* motive-views.

3.e.c.5 Test workshop conclusions

However, all in all our test workshops did not provide us with any insights that drastically changed the direction of the program; there was no particular feature that the children craved or disliked. Rather we felt we had confirmed that the program was on the right track, so long as the workshops had good teachers.

3.e.d The Concert Approaches

But the Toy Symphony workshops we were worrying about were, at this stage, still very ill-defined. Where would the children come from? How many would there be? Would they work alone or in groups? We figured anywhere

from 5 to 20 seemed reasonable, but this appeared to depend mostly on the availability of computers for them all.

Were the computers available? Elementary schools tend to have poor computers, if any, and we were unwilling to plan for a tour of concerts with the problem of having to scrounge spare computers from schools in each city.

Therefore, we would have to bring our own computers with us, and we proceeded to buy eight identical almost-top-of-the-line computers that were destined to tour with us around Europe.

This was also the point at which we finally requested pen-tablet-based input, and a set of Wacom Cintiq [Wacom, 2001] pen-tablet displays were purchased, which would permit our children to draw their compositions directly on the screen in front of them.

3.f The final "*Hyce*" Version

Completed just in time for the first concert in Berlin, February 2002, the final version of Hyperscore added a number of features while passing the main remaining hurdle: Enabling the user to create pieces that were three minutes long.

Visually the main change in this version was enabling the user to pan around a vast screenspace inside which the motive views and the stroke views of the previous version could be placed, along with the capability to zoom to various scales, looking at the strokes closer than had previously been possible or zooming out to observe on a single screen a view that would have taken sixteen screenfulls in the earlier version.

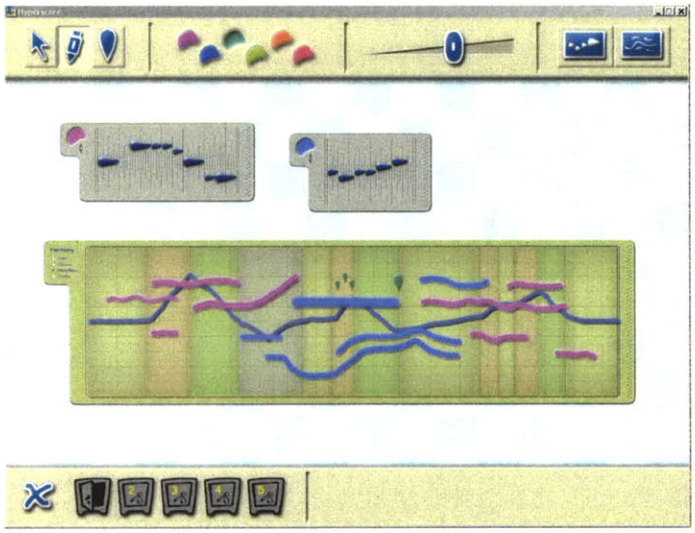


Figure 20: The "Hyce" version. A zoomable interface allows for arbitrary length compositions and motives, unlimited motives, and unlimited space for expansion. Volume and timbre are individually controllable in each annotation. Time flows from left to right, and the spine remains as a way to specify "red" and "green" chord-sequence-variation sections, and "gray" keychange modulations.

This version also added flexibility in length -- all pieces designed in the previous Hyco version had been exactly the same duration. (As a multiple of tempo that is, which could be set by a slider. Pieces with fast tempo completed in less time than pieces with slow tempo.) This version allowed the user to directly stretch both the stroke view and the motive view windows, allowing him to create pieces up to four or five minutes long, composed of motives with any length whatsoever.

This version also contained more sophisticated harmonization and motive-warping algorithms, courtesy of Farbood, as well as the ability to change keys. In all previous versions, although the spine had controlled the chord progressions, the piece had remained in the same key throughout -- *A-minor*. However, with this version for the first time the user had the ability to draw a special symbol, a "spiky" shape, into the spine to signal a key change, with the vertical position of the spike's vertex determining the target key.

The spine was changed most of all, and finally stopped being a free-floating stroke that an unruly child could draw as a spiral, and had become a fixed function that stretched across each horizontal stroke window. We stopped calling it the spine and chose instead the *harmony line*, at that was its only remaining function.

This version also added cut and paste functionality, the important ability to duplicate motives, and an undo key.

3.f.a The Website - The Hyperscore Showcase

Another new feature of the final version concerned a change to the file format so as to support a website, which had been designed in the first months of 2002 to allow more people to learn about our work.

Figure 21: Web surfers may download Hyperscore from the Hyperscore Showcase, a part of the Toy Symphony site at http://www.toysymphony.net.

Hyperscore is unique as a project in the Music of the Future group in the ease with which it could be distributed. The Beatbugs, the Shapers, and the Hyperviolin Bow are all, well, physical things, which are expensive to construct and fragile and hard to ship and subject to all the material inconveniences of a physical devices. The only way they cam be brought closer to the interested public is to post

news and videos on our website. However, Hyperscore, as pure software, can run on any computer whose spec more or less matches the workshop computers upon which it was designed to run. It could be posted on the web in its entirety and anyone could download it, so this presented a fabulous opportunity for our group to have a greater impact on the world.

We aimed to do more than just offer the program for download; rather we were interested in building an "online community" around people using Hyperscore and sharing their creations online. Users could post their pieces, listen to pieces composed by others, and even exchange composition fragments with each other. With these goals in mind it was decided that a bulletin-board web site would be set up and a pair of web designers, Brian and Colin Knep, were hired to make this a reality.

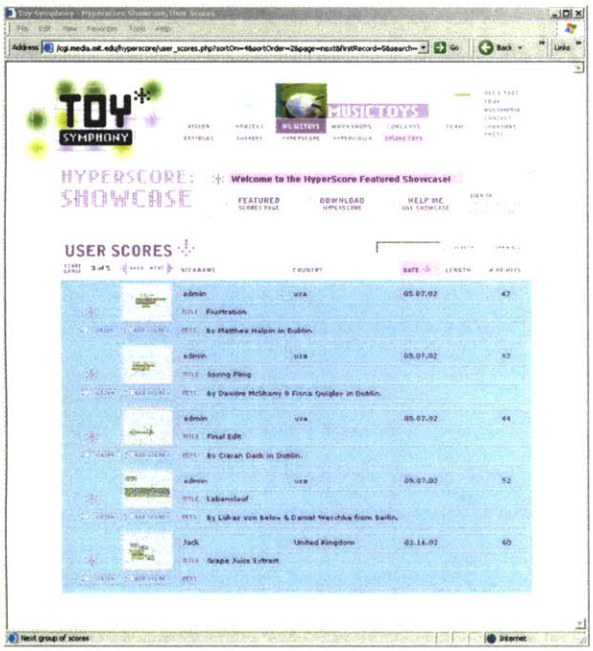


Figure 22: After downloading Hyperscore, users may browse a large repository of existing Hyperscore pieces.

The website contains a login system, and allows users to create accounts and specify passwords for themselves. Once logged in, a user is allowed to upload their own Hyperscore pieces with an attached description, as well as edit or remove pieces they had previously uploaded.

Users just visiting the site, without logging in, are able to browse or download any Hyperscore piece submitted into the database, sorting them by date submitted, length, name or country of origin.

Each submitted Hyperscore piece is presented alongside a thumbnail of the piece and a button to download a MIDI file corresponding to the main piece in playback. One technical issue was arranging the method by which the website acquires this thumbnail image.

As the webserver for the Hyperscore Showcase site ran on a Unix box, it was not feasible to have an instance of Hyperscore running on the server in order to produce MIDI and GIF versions of submitted files. It was therefore decided that Hyperscore would have to save a thumbnail and a MIDI stream into its saved format every time. Thus, whenever a user saves a Hyperscore file, this version embeds into the file a screencapture of the window, and a

MIDI playback corresponding to the motive window or stroke window selected at the time of the save, just so that the website software would be able to extract these should the user happen to upload his piece.

Since being set up in February, the website continues to slowly grow in popularity.

3.f.b DirectX Issues

One troublesome issue involved the fact that the "Hyce" version made use of the 3D acceleration present in many graphics cards to enable the zooming and panning of infinite canvas Hyperscore now supported. However, there are many people in the world whose computers have no 3D acceleration and therefore are not able to run this version, and shortly after releasing the website we began getting a steady stream of emails from people who expected it to work on their old home computers and were dismayed they lacked required hardware.

3.f.b.1 Installing on the Carl-Orff-Grundschule machines

In fact, this problem came up in many other places, including the first time we tried to install Hyperscore on the computers of the Carl-Orff-Grundschule. In mid-January with the Hyce version still in development, we took a trip to Berlin to meet with the teachers who would be working with the Hyperscore children.

The children had yet to be chosen, but in the months while Mary and I had been completing the program, our Toy Symphony producer Ariane Martins had succeeded in finding a German elementary school willing to donate children, a group of music teachers willing to teach the workshops, and buildings and times and people were being smoothly brought together. But during our Test Workshops (see 3.e.c) we had recommended that the children get to work with Hyperscore before the workshop week began, and thus it was up to us to install Hyperscore on whatever random elementary school computers could be found in Germany, because our eight official workshop machines would not be ready until the workshop week itself.

However, when we traveled to this distant school we found their computers to be slow P200 Windows-95 machines with no 3D acceleration at all. Unable to run Hyce without additional hardware, we had to quickly order a dozen graphics cards at $25 each to install the program successfully. The computers worked fine after this.

3.f.c Implementation Section Conclusion

This chapter has detailed the development process of Hyperscore from its initial conception to a working, attractive composition tool. In February of 2002 it was taken to Berlin and used to compose string orchestra pieces as intended, and this worked very well and many of the pieces composed were quite moving.

The program is available for download on the web, and the following chapter will describe the program itself in more detail.

Chapter 4: Implementation

The goal of this section is to describe the final Hyperscore version in detail, including the algorithms used and the rationale for their design. As Hyperscore is freely downloadable from http://www.toysymphony.net, the reader may choose to download a copy and exercise these features as they are described.

4.a Overview

Hyperscore presents a vast canvas upon which the user builds his composition by arranging various graphical elements on a two dimensional space. The screenshot below shows a short piece at F2-zoom level showing a harmonized piece built out of three motives. When a user begins creating

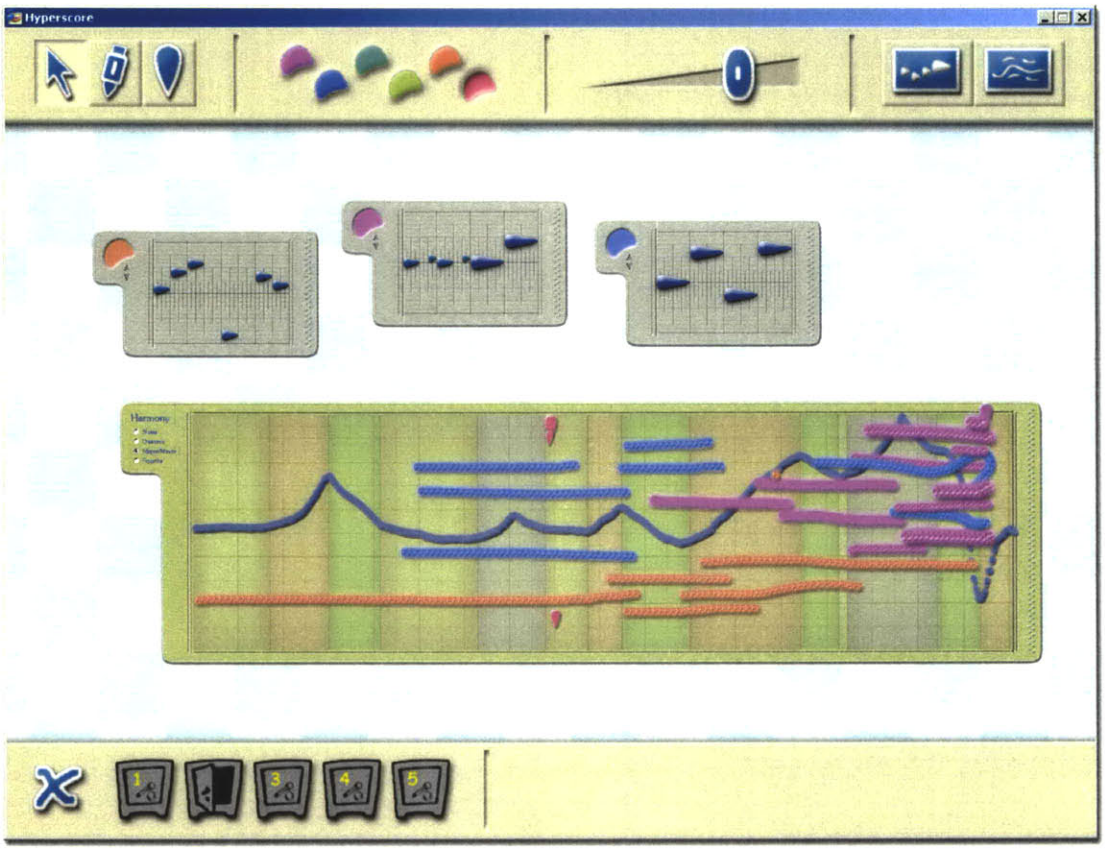


Figure 23: A typical screenshot of Hyperscore. Upon the blue and white checkerboard canvas we see three Motive Windows for three colors, and a Stroke Window inside which these motives are layered upon one another. The red/green background shading denotes harmonic changes.

music in Hyperscore, he is presented with a blank blue-and-white checkerboard canvas upon onto which he can place material:

First the user creates short melodic phrases called **motives**, binding these motives to different colored pens.
Second, the user draws patterns of **strokes** using these colored pens to layer multiple motives together. Each stroke adds an additional voice playing its corresponding motive at the corresponding time and pitch.
Third, the user manipulates a special **harmony-line** to specify key changes and chord variation. The notes defined by the colored strokes are fit to a chord progression based on the harmony-line.

Users generally begin by taking these steps in order, first creating motivic material, then building a multi-voiced texture out of motives, and finally adding harmonization. Following this, users iteratively edit their compositions by adding or changing these three components of a Hyperscore composition: motives, strokes, and the harmony line.

4.a.a. Extremes of scale

The philosophy behind the program is that the process of composition is interesting and creative at two "extremes of scale".

The "smallest scale" information in a composition consists of the individual note patterns, the minute contours and rhythms that make up a piece. By creating Motive Windows the user has direct control of these aspects of his piece. It's very easy for a child to invent and enter simple contours and rhythms, and this permits a lot of creativity.

The "highest scale" information in a piece consists of the overall patterns with which motives blend. A section is composed of interactive motives; the section introduces a third motive and rises to a climax. These overall, long-duration trends in a piece define how it flows and changes over time, and this is the kind of information the user specifies in the Stroke Windows.

However, in between these highest-scale and lowest-scale concerns are a medley of mechanical details. These details include, how to adjust the notes so that they are in the correct key and form perceptible chords, yet maintain the motive's contour. Or, what specific chord sequences will sound good, as well as provide the desired overall trends. This *mechanical* work is one of the things that makes composition so hard; the need to accomplish this mechanical work correctly is why potential composers must be extremely facile at reading and writing in the staff-notation, so that they can position

notes in specific relationships with respect to other notes around it, above or below, in the past of the future.

Hyperscore endeavors to automate this work, allowing children to focus on the more creative aspects of composition, the "highest" and "lowest" levels described above.

4.b Motive Windows

By creating a "Motive Window", the user defines a short, single-voice sequence of notes called a motive, and associates this motive with the color displayed. Each motive window is overlayed with a time-versus-frequency grid onto which the user can place little droplet-shaped figures of various sizes to represent notes.

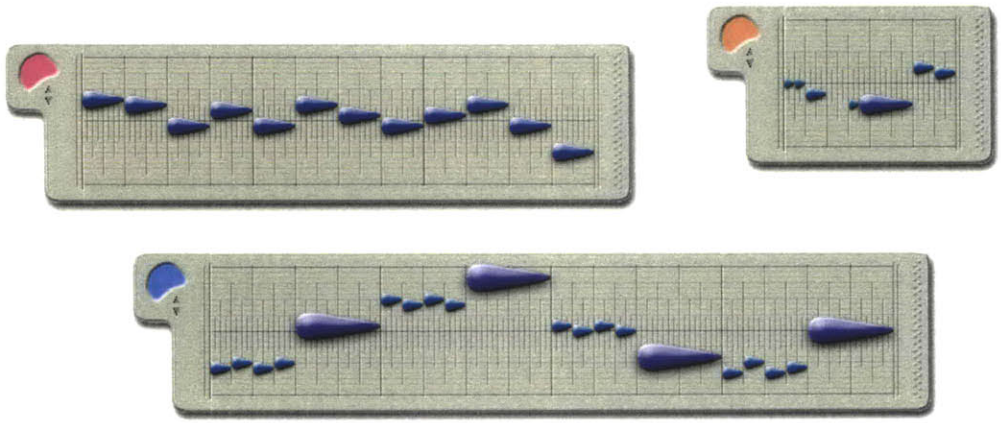


Figure 24: Three Motive Windows associate motives with the colors red, orange, and blue. Motives can be different lengths, can contain notes of different lengths, and can contain spaces (silent rests). But only one note may be present at each horizontal position.

4.b.a. Timing

The horizontal time axis is decorated with a ruler pattern dividing the horizontal direction into "measures" and binary subdivisions of this; the user can stretch the motive's length from (1/4) measure to 5 measures. By adjusting the tempo, the user can change how fast the motives play. With the tempo slider at maximum, a measure plays in 0.67 seconds, at minimum it takes 2.98 seconds to play a measure. However these limits are completely arbitrary, and were selected by making a reasonable guess as to the fastest or slowest piece users were likely to desire.

Hyperscore is not internally aware of time signature; the divisions are purely for the visual convenience of the user. Music frequently sounds better if notes or features occur at regularly spaced intervals, and the alternating prominence of the time markings make it easier to position notes at major boundaries with consistent timing.

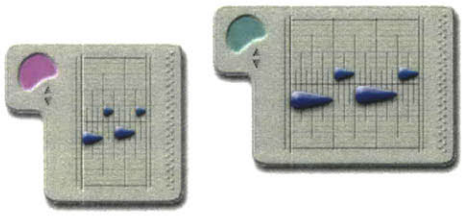


Figure 25: By adjusting the tempo, the user can make either of these motives sound like the other.

Lines divide a measure into 32 intervals, but the smallest note available is the 1/16th note. The largest note available is the whole note. Note durations are provided in increments of (1/16)th of a measure, so the user can create notes of off-binary duration such as (7/16)th of a measure or (13/16)th of a measure, in addition to quarter notes (4/16)th measure or half notes (8/16)th measure. Currently, it is not possible to break a measure into non-binary intervals such as (1/3)rd or (1/5)th measure.

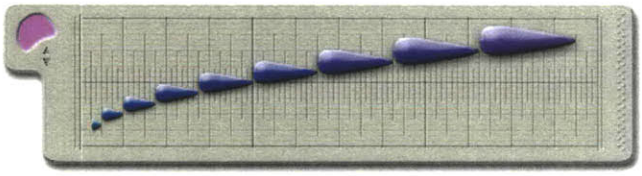


Figure 26: Notes can have many different durations.

Again, the majority of these limits were selected as a compromise between functionality and ease of use. Notes smaller than (1/16)th or longer than whole could have been easily provided, but would have added complexity without much benefit (few traditional songs require both extremely long and extremely short notes). Adding the ability to create triplets would have enabled some interesting rhythms at the expense of more interface elements with which our target age group might have trouble. Simplicity was a goal in the design.

Users can adjust the duration of new notes with the arrow keys. However, while the left/right arrows add or subtract a (1/16)th duration, the vertical arrows multiply or divide by two. Thus a user can switch between commonly used half, quarter, eighth, and sixteenth notes with few keypresses, while all the durations in between remain available.

4.b.b. Pitch

The vertical pitch axis is decorated with evenly spaced horizontal lines each denoting a half-step of pitch change. The central line is a darker color, as are the lines one octave up and down from there. There is room on the window

for two more half steps in either direction, making for a total range of 28 half steps, or two and a third octaves. This range was chosen because it seemed to offer as much as was needed within a reasonable window size. Many motives have a rising or falling contour, but it is rare to find a motive with more than two octaves of difference between its lowest and highest notes.

The window does not restrict the user to the pitches of a single key, nor are the lines corresponding to scale tones darkened or highlighted in any way. The user should not be concerned with the absolute pitches of the notes in a Motive Window, since each note's pitch will be altered when instantiated in a Stroke Window, first by the shape and location of the stroke that instantiated it, and second by the harmonization algorithm.

Therefore, while the timing is copied exactly from the Motive Window to the final score, the pitch information is treated only as a guide, and the user drawing a motive in the Motive Window should take this into account. In addition, as the program is targeted toward musically untrained users, children using this program are not likely to see anything unusual in a smoothly varying set of pitches.

4.c Stroke Windows

After defining motives, the user creates a second type of window on the canvas called the "Stroke Window", a yellow rectangle inside which the user can draw pen strokes of various colors. As in the Motive-Windows, the Stroke-Windows define a space interpreted as a time-versus-frequency graph. However, instead of fixing stroke onto a fixed grid, the user can paint freely.

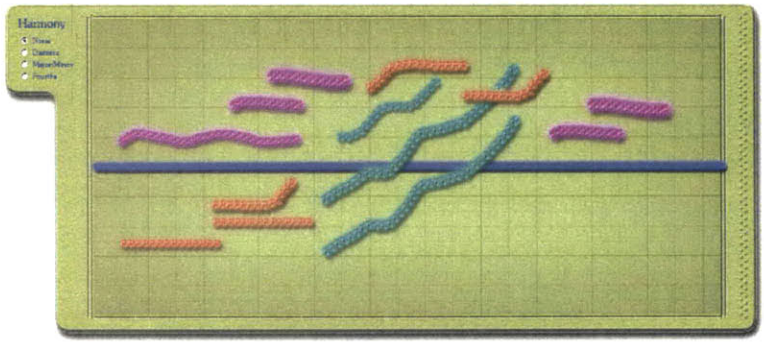


Figure 27: A Stroke Window containing some strokes.

Each pen stroke adds a voice to the composition by taking the motive corresponding to the pen's color, and looping it along the stroke's horizontal length. The vertical variations of a stroke modulate the contour of the motive, allowing the user to reshape the contour of a motive in addition to placing it precisely in both pitch and time.

Note that, somewhere on the canvas the user must have created Motive Windows corresponding to the colors. The figure displays a Stroke Window with pink, orange, and green strokes. However, these strokes are meaningless (silent) unless the user also creates Motive Windows for these colors.

4.c.a Drawing Strokes

Strokes are created when the user draws them with the pen tool. Immediately, the user can play his piece to hear how the new notes blend with what was there before.

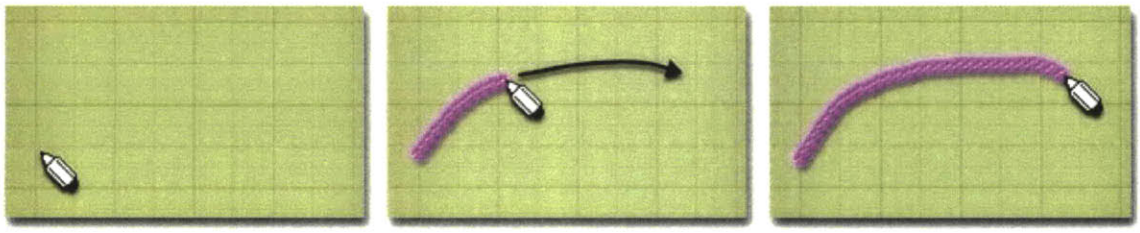


Figure 28: The user can draw freehand strokes upon the canvas.

The stroke is represented as a simple polyline; an array of (x,y) points each approximately 10 pixels apart. When the user selects the pen tool and begins drawing, the first point is laid down. As the user moves the mouse, it's distance from the last point is measured and when it exceeds R pixels, additional points are laid down from the previous dot towards the current mouse position. This allows the user to create a curve consisting of uniformly spaced points.

Ideally, the zoomable interface of Hyperscore would argue for a multiresolution representation of strokes, as in [Finkelstein, 1994]; and this is kept in mind for the future. These representations store long smooth sections very efficiently, concentrating data, precision, and processing power, on areas of a curve where details are numerous. A multiresolution representation would also be the appropriate storage form for the program to display strokes at different levels of detail, an essential feature in zoomable user interfaces. [Perlin, 1996]

However, the polyline representation is an easily implemented form that is most appropriate for the physical simulation, described below, that is used to allow curves to be easily reshaped.

4.c.b Reshaping Strokes

After creation, a user can iteratively edit his stroke by moving or reshaping it, then After each edit, the user listens to determine

In general an intuitive method of editing curves is not widely agreed upon across the various software packages that enable curve editing. Professional drawing packages like Illustrator, et al, represent curves as linked bezier segments, and the user is allowed to explicitly click-and-drag the vertices of the beziers' control polygons.

We intended Hyperscore to be intuitive and easy-to-use even for very young children who might find control-point curve editing tedious or slow. Instead, one is able to simply grab the curve and pull. By using physical simulation, the curve responds in a natural manner, and the child can make small or large changes just like dragging a rope.

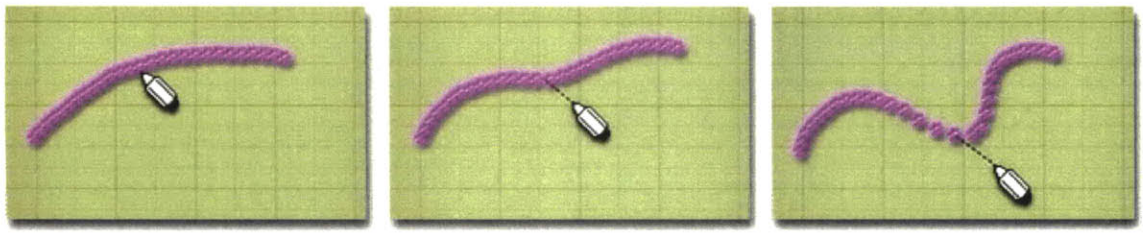


Figure 29: Once drawn, strokes can be reshaped dynamically. The degree to which the curve responds depends on the speed and flourish with which the user moves the mouse. Curves can also be moved without reshaping.

When the user draws an annotation onto a Stroke Window to instantiate a motive, the shape of the motive warps the motive's contour, altering its melody. The user can listen to this, and modify the curve shape iteratively until the he is satisfied. In this interface, modifying the curve is accomplished simply by right-clicking anywhere on the curve's length and dragging. The curve behaves like a physical string with mass and tension, and responds to mouse motions in a realistic manner.

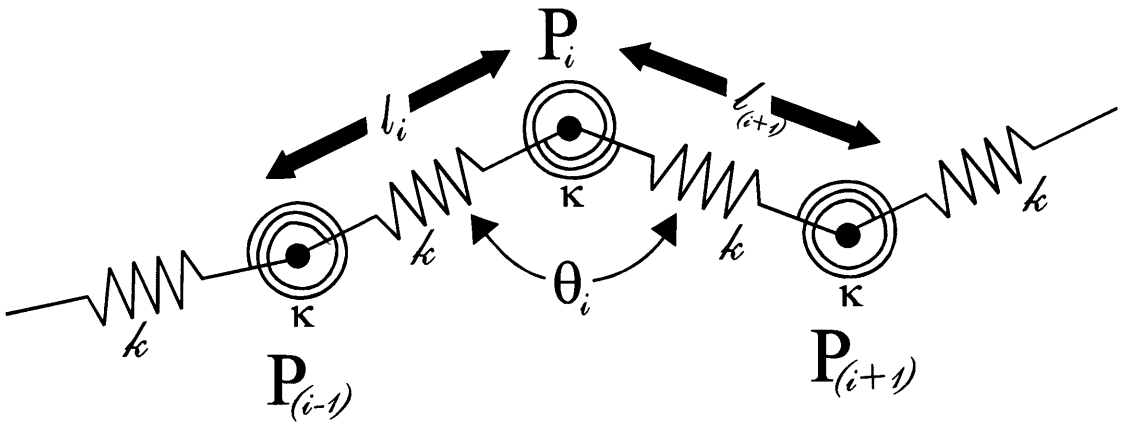
4.c.b.1 Physical Simulation

This dynamic effect of the stroke's physical-seeming response to mouse dragging is implemented as a mass-spring model. Each annotation curve in Hyperscore is stored internally as a list of connected 2D, real-valued, control points *(x,y)*. Each control point is the same distance from the next, usually a few pixels.

The line segment between each pair of control point mass is modeled as a spring with a fairly stiff spring constant (k). Each internal joint (each control point that is not the first nor the last one) is also the site of an angular spring (κ) that acts to keep the angle at that joint around a certain value.

When the user rightclicks on a mass on the curve and begins dragging, a spring is created between the grabbed mass and the mouse position, and a physical simulation begins whereby the mouse's motion, through the mass-mouse spring, passes energy and transfers velocity onto the curve.

The simulation is done in the standard way of physical systems: The state of the system consists of control points with position and velocity. Thus control point i has four quantities $\{ x_i, y_i, vx_i, vy_i \}$ associated with it, and a stroke with N masses has a state vector with 4N quantities. At any specific state, the linear and angular springs exert forces on the masses based on the amount they're stretched, in addition, each mass feels a drag force proportional to its velocity. By Newton's law, these forces correspond to the derivatives of the vx_i and vy_i state components. Thus, if $\boldsymbol{S}$ represents the state vector of a given stroke during the simulation, code totaling the force applied to each mass can compute $d\boldsymbol{S}/dt$.



This function, $d\boldsymbol{S}/dt = F(\boldsymbol{S})$, is integrated across time using a standard Runga-Kutta 4-th-order ODE solver from Numerical Recipes, to simulate the behavior of the curve from each frame until the next frame 1/30th of a second later. [Numerical Recipes, 1992]

In addition, at each frame (every 1/30th of a second), the "quiescent" angle of each of the angular springs, as well as the quiescent length of the linear springs, is altered towards their current angle and length. Thus, if the curve is bent and held for a time with the mouse, upon release the curve will have "solidified" towards the state at which it was held.

Currently, the angles and lengths are altered towards their current value by 5 percent. (NewQuiescent = 95% OldQuiescent + 5% CurrentValue) This quantity, as well as the precise values assigned to the stroke node masses, the stroke's linear and angular spring constants, the stroke-to-mouse spring constant, and the damping force coefficient, were all chosen arbitrarily by iteratively varying them until smooth, responsive dynamics had been obtained.

Curves which behave in this manner can also be seen in the work of Golan Levin. [Levin, 2000]

4.c.c Strokes' Effect on Motive Contour

Each stroke adds notes from its motive to the piece based on stroke's position, curve, and length. The following section will describe the manner in which a stroke modifies the notes of a Motive and adds them into the piece. However, these are not the final notes heard by the user unless the "Harmonization" feature is turned off. In the more common case that a harmonization algorithm is active, most notes placed by the strokes are later adjusted in pitch to fit a key or chord progression.

To precisely understand the manner in which the strokes affect the notes, it is helpful to consider the horizontal and vertical sections independently.

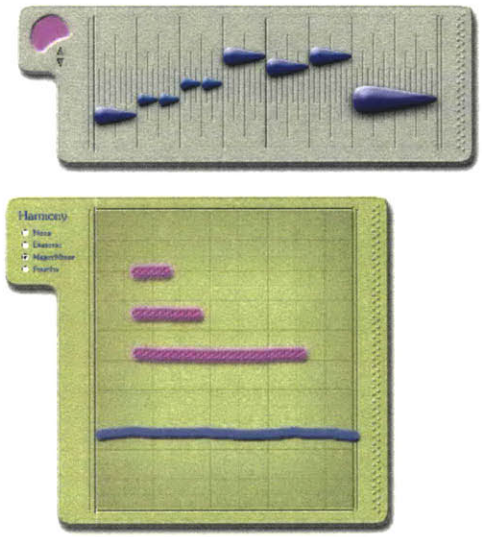


Figure 30: Given the above fragments, the top red mark sounds only the first five notes of the motive. The middle stroke plays the entire motive once. The bottom stroke plays the motive more than twice. How much real time this actually takes depends on the setting of the tempo slider.

4.c.c.1 Time

In the horizontal, temporal direction the rhythmic patterns of a stroke's motive are reproduced exactly. The motive begins playing (or rather, its notes are inserted into the pre-harmonization score) at the time corresponding to the horizontal position of the leftmost stroke endpoint. The motive plays in a loop continuously until the time corresponding to the horizontal position of the rightmost stroke endpoint.

If the stroke's horizontal extent corresponds to an uneven number of

motives, the final motive is cut short. Indeed, if the stroke's horizontal extent corresponds to less than a single play-through, only the first part of the motive will play.

Although both Motive Windows and Stroke Windows are time-vs-frequency images, the time scale used in the Stroke Windows is six times denser than that used in the Motive Windows. Therefore, to play a given motive once, the user would draw a stroke whose horizontal extent on the screen was one sixth the width of the motive itself.

One of the consequences of this behavior, unfortunately, is that the user can have trouble drawing a stroke to play a motive a fixed number of times because there is no visible mark at the repetition point. Another difficulty lies in aligning one stroke to begin playing at the same time as a specific note in another stroke. Solutions to both of these problems, such as automatically "snapping" a curve horizontally to align with previously placed curves, have been addressed as necessary improvements for the future, and were not included in the current version because of development time constraints.

4.c.c.2 Pitch

While the horizontal position and extent of a stroke determines at what time, and how many times, to play the notes of the motive, the pitches of the notes played are affected by vertical variations in the stroke. The general idea of the reshaping algorithm is that the motive is reshaped as if the pitch of the Motive-Window's central line followed the contour of the stroke up or down in pitch. However, a motive's notes will not shift so far as to destroy the rising and falling pattern of the Motive.

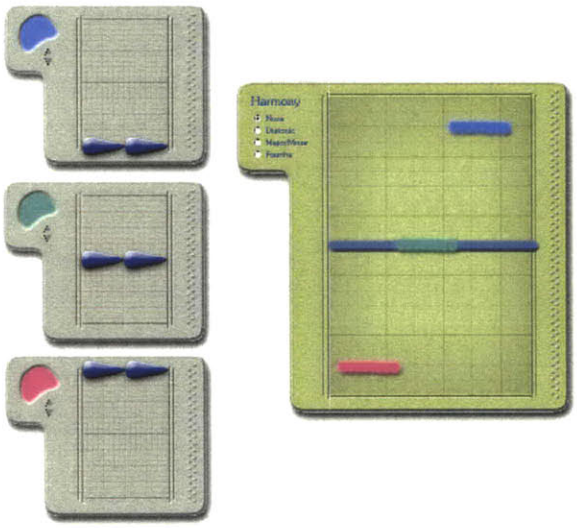


Figure 31: This Stroke Window actually plays the *same note* repeatedly. The notes of the red motive, one octave high, are transposed an octave *down* by the low placement of the red stroke. Similarly the low notes of the blue motive are transposed *up* by the high placement of the blue stroke. All notes end up with the same pitch.

The simplest stroke contour is one that is perfectly horizontal. In this case the motive is simply transposed in pitch, the center line of the Motive Window is set to the pitch corresponding to be the vertical position of the stroke. Very simply, moving a stroke higher causes the motive to play with higher notes, while moving it lower transposes down instead.

The Stroke Window's available pitch range is similar to that of the motive windows: a little more than two octaves. The strokes in the figure show the vertical positions that affect a one-octave transposition. The highest notes possible in Hyperscore occur when a motive with very high notes is instantiated by a stroke that is itself very high on the page. The opposite applies for the lowest notes.

If the stroke contour is bent or curved, the degree of transposition varies along the length of the motive and the stroke's contour is blended with the motive's own contour. However, the system will not allow the rising and falling patterns of a motive to be altered by a stroke. For example, it is not possible for a contour to transform an ascending sequence of pitches to a descending one. If a motive consisting of a note followed by a higher note, is instantiated by a rapidly descending stroke that aims to pull later notes down lower than earlier notes, the most the second note may descend is to the level of the earlier note.

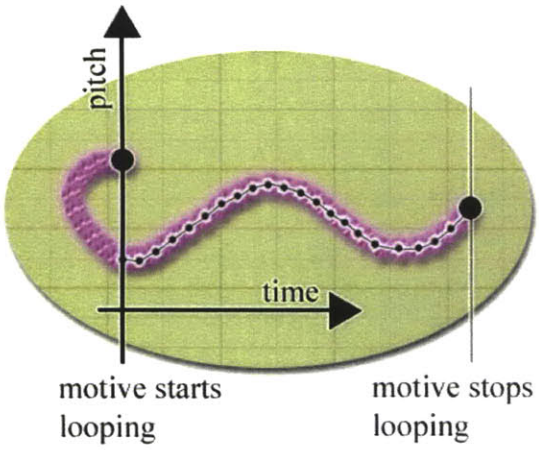


Figure 32: Areas of a stroke outside the interval between the two endpoints are

Thus the shape of a stroke may make an ascending sequence more or less ascending, and in the limit flat, but may never become a descending sequence. The opposite applies to descending sequences. However, flat sequences may become ascending or descending without restriction. If the user creates a completely flat motive with only rhythmic information, the user's strokes completely determine the motive's contour.

The motive-reshaping algorithm was designed and implemented by Mary Farbood and more information can be found in her thesis. [Farbood, 2001]

As is clear from the figure, the user's strokes are intended to be functions with single values between the two endpoints. Thus it serves no purpose in this version to draw circles or self-intersecting shapes that fold back upon themselves. However, the unrestricted nature of the canvas contributes to the open, creative feel of the application. Minor loopbacks and vertical sections, as might easily occur as a part of any freehand gesture, are tolerated and this makes for a more enjoyable experience.

4.c.d Volume, Timbre, and the Melody Flag

In addition to its position and shape, each stroke has several other properties that affect the sound of the notes it contributes to the piece.

First, a user can control the volume of each stroke, and this parameter corresponds to the thickness with which it is drawn. The thinnest and thickest strokes possible range from very quiet to quite loud. However, it is not possible to draw a stroke whose thickness varies along its length.

As the program was designed to be controlled using a pen-tablet, the pen's pressure could easily have been mapped to stroke thickness, allowing the user to vary thickness along a stroke. However, realizing this mapping also had the consequence that it became very difficult to draw a stroke which began at full volume. When drawing a stroke with a pen, it is natural to apply light pressure at the beginning where the pen first touches the tablet, however, it does not always sound good to have each motive fade in from a low volume. Naively correcting this defect by hardcoding that a stroke's beginning must be at full volume would have had problems as well; it would then have been difficult to create a stroke entirely at low volume.

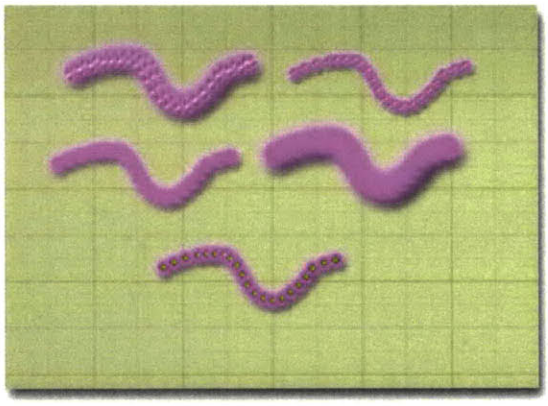


Figure 33: Similar strokes with different settings for thickness (volume) and texture (timbre). The bottommost stroke has the melody flag enabled.

Many interface solutions can be conceived to offer this control to the user, but doing so in a way that maintains the program's simplicity remains a task for the future.

A second control is a timbre flag, visible as the presence or absence of a course stipple pattern on the stroke's surface. A stippled stroke will be played with a plucked-string sound, a smooth stroke with a bowed-string sound. No other timbres were needed because Hyperscore was designed explicitly for children to compose for a string orchestra, where strings are the only sounds available.

A third control is a *melody* flag, rendered as a series of plus signs. The melody flag, if enabled, affects the method by which the notes are harmonized. Harmonization, discussed in section 4.d below, is a process by

which notes are adjusted in pitch from the values given them by each stroke's position and shape. The melody flag has no effect if harmonization is turned off. However, if turned on, notes which would otherwise be adjusted to fit the current chord in a chord sequence, are instead adjusted merely to be within the current key.

The result is that the notes, corresponding to a stroke with this flag engaged, may sound more prominent, because they are less likely to become obscured by other notes playing simultaneously. The read should consult Farbood's thesis [Farbood, 2001] for specifics of the note-adjustment algorithm.

4.d Chord Droplets

In addition to pen strokes, individual Stroke Windows may also contain droplet-shaped objects which produce chords.

The chord shapes are similar to strokes, in that their function is to insert notes at a given time and base frequency. Each droplet schedules three notes to play at the same time, separated by intervals of pitch that depend on the droplet color. The different colors of droplet introduce different kinds of chords, with different intervals between their three notes.

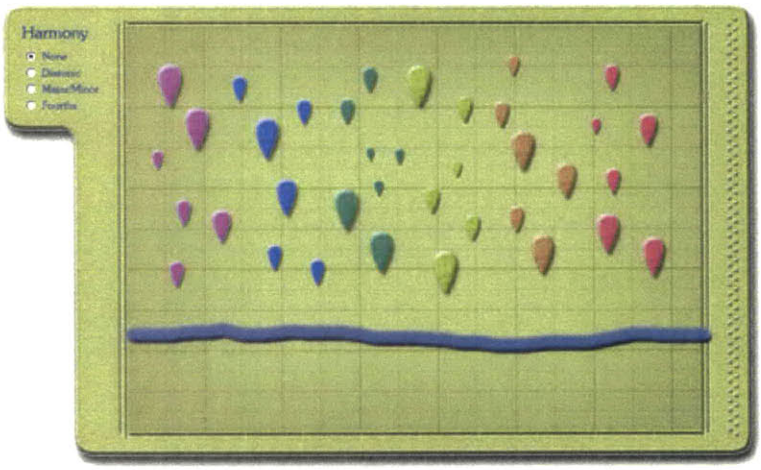


Figure 34: A Stroke Window containing some chord droplets. The various sizes correspond to different volumes, as with strokes.

However, the notes introduced by droplets are just as subject to later harmonization as notes introduced by strokes; therefore the mapping between droplet color and chord type is relevant only where harmonization is *disabled*. With harmonization turned on, simultaneous notes introduced by droplets are likely to be adjusted up or down in pitch differently, changing the intervals between the chord notes.

Chord notes differ from stroke notes in their timbre; chord notes sound as if played by a harp.

4.e Harmony Line

The user can assemble an intricate textured collection of notes by placing strokes in a Stroke Window, but without the application of a harmonization algorithm these notes are likely to be completely atonal. The strokes as presented so far treat all semitones equally.

If a composer is aiming for an atonal piece this may be exactly what is wanted. However, users aiming for more traditional music may activate a harmonization algorithm in a Stroke Window. With this engaged, notes positioned by the user's strokes and chord symbols are adjusted in pitch to match a chord progression that the user can influence by shaping a "harmony line".

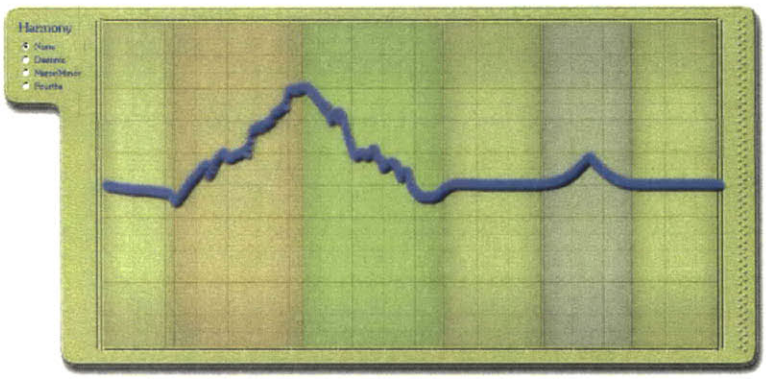


Figure 35: A Stroke Window whose harmony line has been manipulated to produce distinct sections. (Of course, as shown, this window would be silent because it lacks any strokes or chords.)

The harmony line appears as a flat blue line across each Stroke Window, but can be modulated to vary vertically across its length. As the user sculpts peaks and valleys into the line, the program identifies regions which are on the overall rising, or falling, or which contain a smooth concave spike; and these regions are shaded red, green, and gray respectively. These regions affect the key and chord progression used during harmonization.

The red, *ascending* regions induce chord progressions of increasing tension, while green, *descending* regions induce chord progressions of decreasing tension. The user is not given control of the actual chords applied, because it was believed this would be too difficult for people in Hyperscore's target age group. However, the harmony line gives the user overall control of where chord variation occurs.

A gray, *concave spike* region causes a key modulation to occur. During the time corresponding to a gray section, the key is modulated from the key in effect prior to the modulation, to a new key which depends on the vertical position of the spike vertex. The new key then remains in effect to the right

of the gray region. The vertical axis is divided into evenly spaced intervals corresponding to different destination keys, however, they are left unlabelled because people in Hyperscore's target age group are unlikely to know the names of the different keys. Rather, a visual relationship is established: the taller the spike, the further away the target key and the greater the auditory effect of the transition. The harmony line gives the user overall control of key modulations in addition to chord variation.

The local curvature of the harmony line is computed also, and this quantity, a function of horizontal position, is used to select the chords used in the various section. Thus, a red, rising section enclosing a particularly shaky harmony line is likely to use different chords than one enclosing a smooth harmony line. The precise algorithms used to select chords, and to alter notes to fit them, were designed and implemented by Mary Farbood and more information can be found in her thesis. [Farbood, 2001]

4.e.a Harmony Line Editing Behavior

Similar to the annotation stroke model, the harmony line is modeled as a physical object with masses and springs. It responds in a physical manner and the user can edit the harmony line by simply grabbing it. However, the harmony line is not a freely bendable curve but rather is constrained to being a pure function, thus variation is allowed in height only. Each control point mass is fixed in its horizontal location.

As with the annotation strokes, when the mouse grabs a control mass of the harmony line, the program simulates a spring connecting the grabbed-mass and the mouse. By moving the mouse, the spring drags the grabbed-point behind it, which in turn drags its surrounding segment of curve. However, unlike the stroke simulations, the mass-to-mouse spring does not remain connected to one mass, rather it moves horizontally with the mouse. The mass-to-mouse spring is always vertical, connecting the mouse to the mass at its horizontal location.

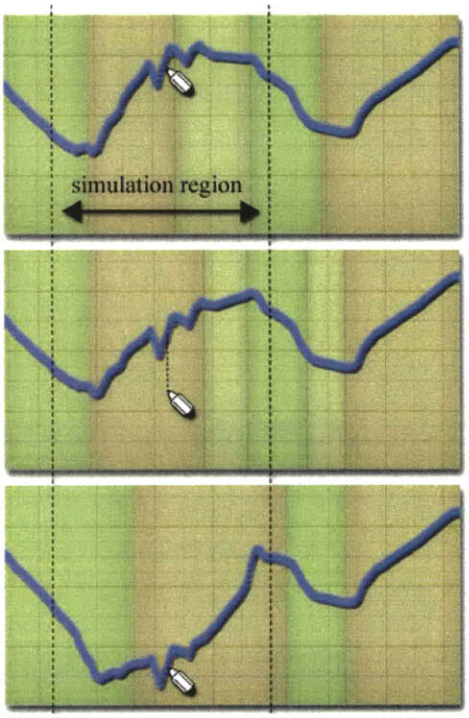


Figure 36: A region of the harmony line surrounding the mouse is physically simulated. The user can smoothly deform regions by grabbing and pulling.

In addition, because the harmony line extends the length of the Stroke Window, for long pieces it can be composed of very many masses that would be slow to simulate all together. Thus the program simulates only the masses within a small *simulation region* around the mouse, 32 masses in either direction. With the masses separated by 10 pixels each, this corresponds to approximately one third the screen width.

However, after testing this interface, complaints were received that it was not possible to reshape with a simple drawing gesture. Users wanted to be able to quickly change the curve to a new shape, while the physical model required the user to coax the line into a desired shape by tugging and pulling. In this case, the physical model appeared to be a detriment. However, the physical model also had the positive feature that it maintained the harmony line's continuity. If the user attempted to create sharp discontinuities in the harmony line, the springs between separated control points would bring them back together.

Therefore, the harmony line simulation adopted a unique algorithm that worked as follows: When the user, after grabbing a point on the harmony line, begins drawing by moving the mouse to the right or left, the simulation region follows the mouse but rapidly *overtakes* it. The physical simulation regions moves in the direction of motion faster than the mouse, until the simulation region is completely ahead of the mouse.

Notice, in following figure, the simulation region is fully to the right of the mouse, whereas in the previous figure, the simulation region was positioned to place the mouse in the center. mouse was in the center of the The left edge of the simulation region becomes connected to the horizontal position of the mouse, moving right. Similarly, if the user begins to draw left, the region slides left as well, until the right edge of the region jams up against the mouse.

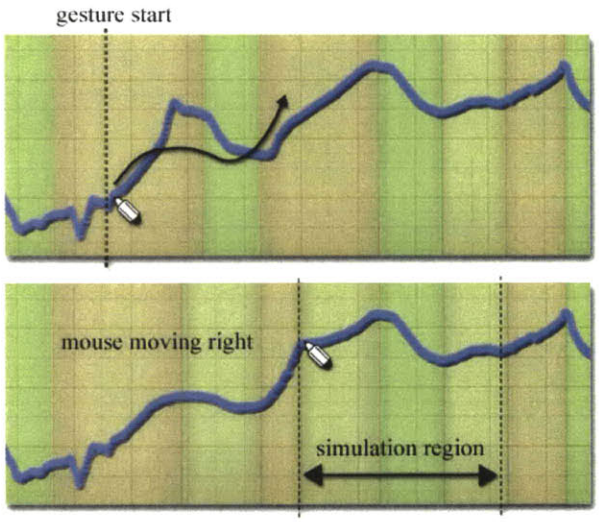


Figure 37: When the mouse moves at significant speed in either direction horizontally, the simulation region slides until it is completely out in front of where the mouse is going.

Thus, while the user is drawing using a rightward moving motion, the harmony line behind the mouse does not animate and acts like a simple drawing operation. However, the harmony line in front of the mouse "jumps up" to maintain continuity.

4.e.b Harmony Line Segmentation Algorithm

As described in Chapter 3, the harmony line evolved during the implementation process from the original idea of a timeline, or "spine" curve. However, its purpose changed across the versions and in this version controls only the key and chord progressions, in two ways. The main purpose of the harmony line is to define specific boundaries between rising, falling, and "spiky" sections that define regions of increasing tension, decreasing tension, and key modulation, respectively. In addition, the overall "bumpiness" of the line texture is used to set the degree of variety of chords selected for the chord progression.

The harmony line segmentation algorithm operates in two passes. First, the line is checked for the spiky regions defining key modulation regions.

4.e.b.1 Seeking Spiky Sections

This is done by first scanning the Harmony line for extrema points: points higher or lower than its neighbors. For all vertex points, the slope of the four points to the left are compared to the slope of the four points to the right. If the two slopes are nearly equal and opposite, and exceed a threshold in magnitude, then the point is considered further.

The final confirmation that the vertex indeed is the centerpoint of a spiky structure is the direction in which the slope changes further from the vertex. If both slopes drop in magnitude as one moves several control-point-masses in either direction of the vertex-mass, then the region around the vertex point is concave, and qualifies as a key-modulation spike. The left and right boundaries of the spike are found by seeking for the points around the vertex where the slope drops to below a second threshold.

This process takes a harmony line and finds zero or more spike-regions. The regions between the spikes, and the region before the first spike and the one after the last spike, are next scanned for rising or falling regions.

4.e.b.2 Scanning Red/Green regions

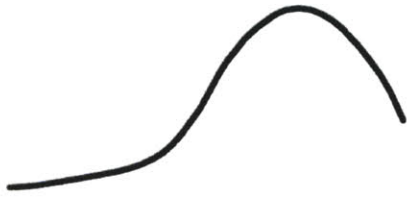
The red/green control regions roughly correspond to rising or falling segments of the harmony line. However, only broad areas of rise or fall should be considered - single pixels out of place may create dozens of tiny rising/falling regions, and these degenerate regions are to be ignored.

Therefore, the scan begins by smoothing the curve several times, by averaging each point with the average of the two surrounding it. The smoothed region is scanned for extrema, which divide the region into several subregions. Next, each subregion is fit with a line that is subtracted off, leaving a residue curve that is, again smoothed and scanned for extrema.

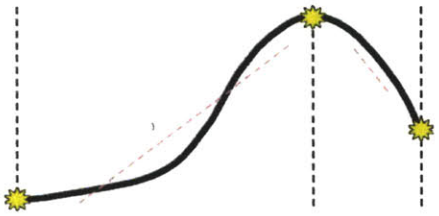
These two extrema scans; the extrema of the original regions, and those in the nonlinear residues of the resulting subregions; both return boundaries that are now treated on an even footing. The intervals inbetween these locations will form the red, green, and clear (flat) regions. Each of the intervals is fit with a line, and its slope thresholded to select one of the three possibilities for each interval.

Finally, the system scans through this list of regions and coalesces or eliminates adjacent identical regions, or small flat regions in between long segments of rising or falling. Also, regions too small in size are eliminated, and their domains distributed among its neighboring regions.

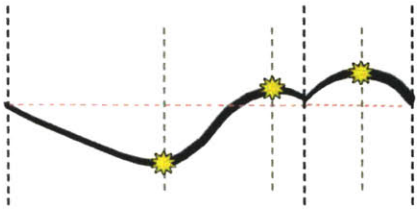
This section list is then passed to the Hyperscore music engine, which uses this information to select a chord sequence, and adjusts all the notes to fit this chord sequence. Details about the harmonization and chord-sequence selection algorithms are to be found in Farbood's thesis [Farbood, 2001].



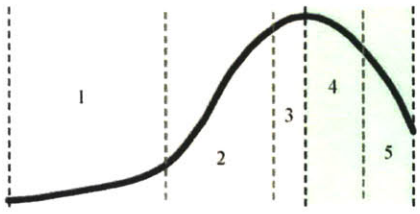
After smoothing, extrema points are isolated, in this case dividing the curve into two fragments.



Each fragment is fit with a line, which is subtracted off.



This permits three more extrema points to be identified. Returning to the original curve, the slope of each section is tagged as rising, falling, or flat.



Adjacent identical sections are merged. Had section 3 been tagged as flat, it would still have been merged with 2 because of its small relative size.

4.f Zooming Graphics

The ability to display fully functional interface components at arbitrary resolutions and to pan and zoom across them is an unusual capability, and implementing this in software would be processor intensive. Previous zooming systems, such as Pad++ and Jazz, decrease processing requirements by dealing only in vector graphics. During a zoom or pan operation, the entire screen surface must be redrawn every frame regardless of the form of the graphic data. With vector graphic data, which tends to consist of simple lines or fills, this is quite feasible. In addition, a page of vector graphics can be resized simply by repositioning the vertices, and requires no more work than panning.

However, the kind of colorful bitmap graphics that we intended to use in the Hyperscore interface would have required the system to perform too many cpu-consuming per-pixel computations, which interfere with the reactivity of the interface. By utilizing the hardware graphics acceleration API, Direct3D, Hyperscore was able to maintain a high level of responsiveness while also benefiting from the attractive look of bitmap graphics.

This was done by instantiating a 3D environment consisting of textured triangles all of which faced an orthogonal camera. In 3D game design, objects constructed out of triangles constrained to face the camera are called "billboards". With lighting and shadowing effects turned off, a graphics card's 3D rendering capability can be harnessed as an engine for rendering rectangular bitmapped regions at any scale.

Chapter 5: Toy Symphony

The goal of this chapter is to describe the Toy Symphony workshops which have provided the several extensive periods of testing that Hyperscore has received so far. The section describes, compares, and contrasts our experiences in Berlin, Dublin, and Glasgow, and includes images of pieces actually produced by musically untrained children.

Toy Symphony was an international music performance and education project that provided the environment within which Hyperscore was tested. Three concerts were held, in Berlin, Dublin and Glasgow, and before each concert a weeklong workshop was organized in which children were to construct three-minute pieces for string orchestra. At the end of each week, the most successful pieces were manually transcribed into standard notation and played at the concerts, giving the young composers the experience of hearing their week's work played live.

5.a Berlin

- **Concert and Open House**

  Date: Sunday, February 24, 2002
  Location: The BFR Radio Auditorium, Berlin, Germany
  Orchestra: The Deutsche Symphony Orchester

- **Hyperscore Workshops**

  Date: Monday, February 18, 2002 thru Friday, February 22, 2002
  Location: The BFR Radio Building, Berlin, Germany
  Children: Carl-Orff-Grundschule elementary students

The Berlin concert on February 24, 2002 was the first Toy Symphony event and the weeklong workshops that preceded it were the first serious use of Hyperscore. While several hour-long test workshops had been held intermittently during development, no children had used the program over a period of days or worked on a composition of significant size. Thus each

milestone approaching the Berlin concert was approached with great care and the awareness that the program was going in untested.

As a consequence, the five days of this first workshop week were each very different learning experiences, as day by day we adjusted our instructions to the children in response to their progress from the previous day. It was in Berlin that we learned how to run a classroom around Hyperscore, and the following two concerts provided only refining experience.

5.a.b Monday, Day 1 *(Motives)*

The workshops began on February 18, 2002. Seven top-of-the-line computers, each equipped with a tablet display, a pair of headphones, and an installation of Hyperscore, were set up in the workshop room awaiting the children. Two children per computer was expected.

Figure 38: Children Anna-Isabell Bergert and Annina Iwanowitsch (left), and Igor Quakatz (right) work on Hyperscore during the Berlin Concert workshops, February 18, 2002.

At the appointed time the fourteen children arrived along with three local teachers, and we began by spending some time introducing ourselves, meeting the children, and getting them organized in the room. The children paired up and sat two to a computer, and immediately began exploring the software. They had used previous versions of the program several times at their local elementary school, thus very little time had to be spent covering the interface. We showed them the newest interface changes they hadn't yet seen, and introduced the children to the goal of making a three-minute long piece by the end of the week, to focus them on the task at hand.

Luckily, the language barrier did not turn out to be a great concern. Whenever we had a suggestion or instruction for the children, or they had a question for us, one of the teachers would come over to translate to or from

German, and this worked remarkably efficiently. In addition, many of the children knew English on their own.

We had decided the first day would be devoted to making motives, so we asked them to spend this first day creating some 3 or 4 motives that they liked. However, most children had already developed motives during their time with previous versions at their school, so in fact the children had several motives they already liked already saved on disk. Some children created new motives, others worked on stroke windows.

For the next two hours, the music teachers, Kevin Jennings, Mary and I circled the room looking over the children's shoulders trying to determine what they were doing and giving suggestions. In general, we were looking for variety, so whenever we'd see a child making many motives that contained only tiny notes, we'd suggest that they add longer ones. Whenever we'd see a child making only motives where the notes went across in straight lines, we'd suggest that they add vertical variation. In general, while we were pleased to see that different children gravitated toward different forms of expression, we wanted each child to be aware of the full space of possibilities.

After the workshop was over and the children had left we carefully listened to everything we found in each child's workspace. In most of the workspaces a single Stroke Window had been left prominent suggesting it was that child's main work, (a few children had worked on more than one Stroke Window). For each we tried to determine what sort of individual attention would be most useful for each child.

5.a.b Tuesday, Day 2 (Texture / Cut and paste)

We began the second day by instructing the children to leave their motives alone and focus now to blending them together in a Stroke Window. We suggested that one method of quickly producing length was to cut-and-paste some favored material several times over, changing some aspect of it each time. This was a fine way of maintaining a balance between familiarity and a sense of motion, as well as an easy way for a child to get his piece close to the required length in a short time. We introduced the harmonization modes and encouraged them to work in the "Major/Minor" mode that sound most like traditional orchestral music.

We had taken two children's works from Monday, the Lebenslauf piece, (which at the time had only one node), and Spaziergang (which consisted of only a the first few curves), and we had expanded them, pointing out interesting features, such as how the lines within Spaziergang connected.

We emphasized being careful about the timing of things, aligning various strokes horizontally.

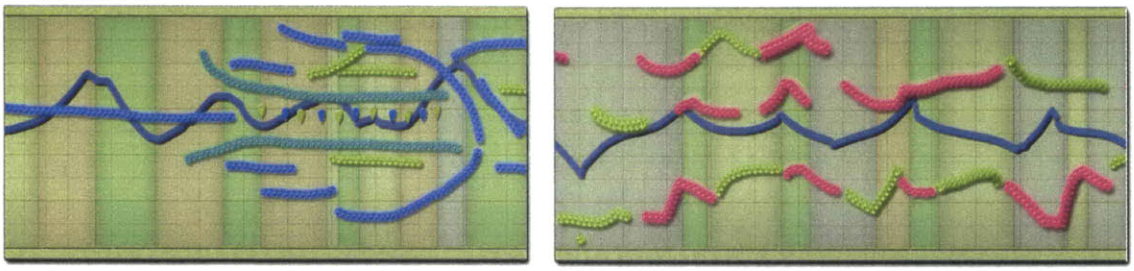


Figure 39: The children who eventually created the Lebenslauf piece had produced the fragment at left by Monday. The rounded structure sounded complex and interesting. We demonstrated that such a piece could be made 3 times longer by duplicating the structure and adding some transitionary strokes. On right is a fragment of the Spaziergang piece, completed on Monday. The child who created this had spontaneously decided to connect red and yellow strokes, end to end, creating a single voice that alternated irregularly between two motives. We praised this and suggested the idea to the rest of the children.

The children were then sent on their way to do their work and we returned to the previous day's pattern of migrating around the room trying to make sure that all the children seemed to be lengthening their pieces successfully.

The main effect that we noticed, afterward as we scanned the children's work again, was that while all sounded fairly good and most children were aligning motives in time as we had asked, there wasn't much variation. The pieces sounded like "texture" without much variety, and would have been boring to hear for three minutes.

5.a.c Wednesday, Day 3 *(Climactic Changes)*

To present to the children methods of preventing monotony, Mary composed the following piece which became a standard part of our later workshops and demos. This piece nicely presents a lot of the different features of Hyperscore in a single work whose complexity ramps up as the piece plays, yet changes slowly enough that the listener can understand what's going on.

It begins with a single motive, playing in a loop. It plays twice as it was written, while the third and fourth repetition takes place in the red and green sections, respectively, so the listener can clearly hear the effect of the red and green sections upon a single motive. Next, additional motives begin playing, increasing the complexity. After the second motive has played twice, the pink motives arrives to introduce the bowed timbre, completely changing the

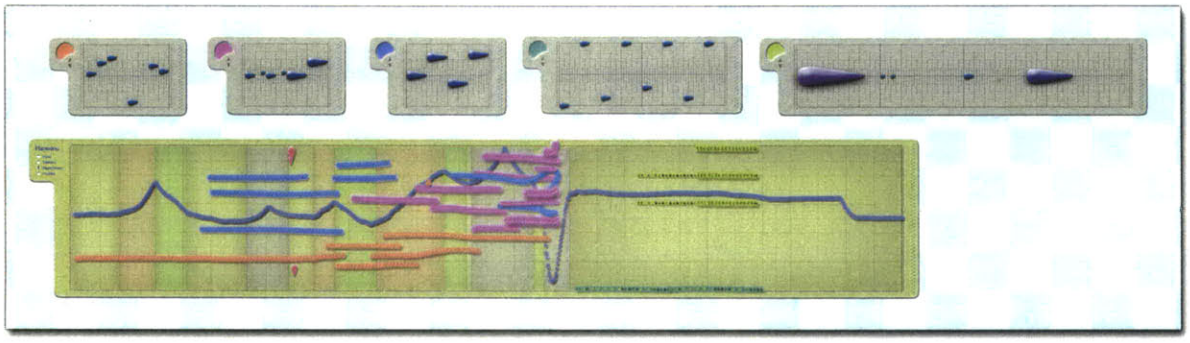


Figure 40: The demo piece called "Contrasts", by Mary Farbood. Observe how the piece gradually introduces the features of Hyperscore.

overall feel. The piece continues to build in tempo and volume reaching a climax that abruptly halts with a measure of rest. And then, a completely new motive begins.

The piece demonstrates the benefit of designing a piece that builds toward a single "goal", or climax, as well as teaching the concept of contrasting sections. The children listened to the piece along with explanations of the different aspects that made it work, and all the children were excited to do something similar. By the end of the day all the children had similar ramping crescendos in their pieces, most notably the one titled Einsamkeit. (See the final pieces in section 5.b.h)

Of the other six pieces, the drumlike rhythms of Igor Quakatz 's piece and the great contrasts in Emilia Schmidt & Anna Walker's *Flohzirkus*, were the most interesting. It was clear at this point that these three were the pieces most worthy of being played in the final concert.

5.a.d Thursday, Day 4 *(Melody Lines)*

By this point all the children had interesting textures with periodic variations and climaxes, but the pieces lacked any single, strong melody in prominence over the other sounds. For this reason we altered the application in the morning to add the "melody" flag (see section 4.c.d), by which any stroke could be set to instruct the harmonization algorithm to exclude its notes from the chord progression.

The "melody" button was introduced to the children, and various motives were played both with and without the "melody" feature engaged until all children agreed that they could hear the difference and liked when a single melody stood out over its background. We encouraged the children to use

this feature, and, though they did so intermittently, it provided another powerful way for the children to introduce variety into their pieces.

By this point we had begun to focus more of our effort on the 3 pieces of highest quality, leaving the remaining four groups more often alone. We no longer assisted all the students evenly, rather, Kevin sat down with Igor and worked by his side for the course of the session, while Mary and I focused on the others.

5.a.e Friday, Day 5 *(Finishing it up)*

By this point we had decided upon the pieces that would be chosen for the concert, and the task awaiting Mary was to arrange and orchestrate these pieces, and print out copies on standard staff notation for the orchestra to receive and rehearse. There was some time pressure at this point, because the concert was scheduled on Sunday, so the orchestra had to have the pieces that they were going to play by Saturday morning in order for them to have a chance to rehearse the pieces.

Up until this point we had believed Mary would only have to transcribe one piece, and realizing that Mary would have to transcribe three made it important to start as soon as possible. The children were encouraged to "clean up" their pieces and perform what final fixes they wanted, but the workshop was over for all practical purposes.

Several children declared that they were going to start new pieces from scratch, and Igor was absent entirely. Kevin spent the final day working on Igor's piece by himself, editing and compressing his work.

5.a.f The Concert

During Friday night Mary and Kevin transcribed the Einsamkeit (Loneliness) piece, the "Changing Seasons" piece, and the Flea Circus piece that had met with Tod's approval. Meanwhile, I assembled graphic posters of each child's piece, and a handout showing the three selected scores that would be printed and passed out to audience members.

During Saturday the orchestras rehearsed the pieces, and on Sunday the concert and open house ran without a hitch. The pieces were moving and beautiful, and it was a real thrill to see our long period of work come to fruition.

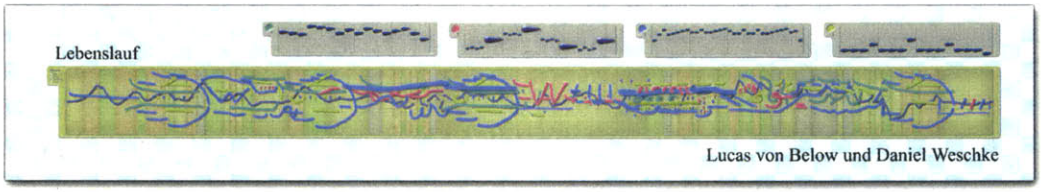
We were very impressed with the work the children had produced in only five days (for they really hadn't made use of any of the content we'd brought over from the elementary schools that they'd composed before the workshops.)

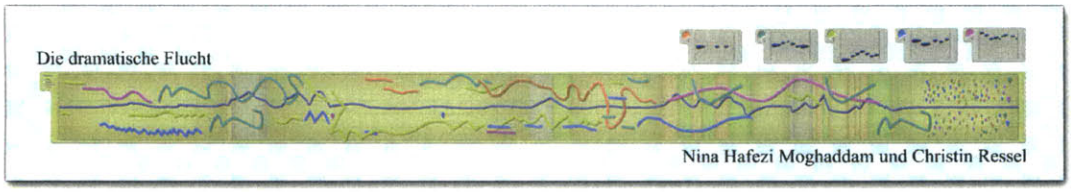
Figure 41: Left: Concertgoers explore the Toy Symphony instruments at the Berlin concert open house. Computers running Hyperscore were available, the demo stations manned by the children who had composed in the workshops. Right: The Deutsche Symphony Orchester prepares to play the children's pieces, along with the rest of Toy Symphony.

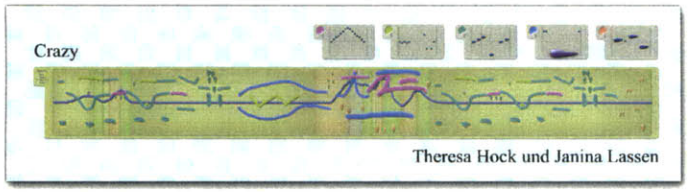
The only problem that was apparent during the concert, was the lack of a projection system made it impossible to project any cursor moving in time for the audience. While all the patrons were given printouts of the Hyperscore scores (see the piece in section 5.a.h), the audience was not given much guidance at the concert about how to read the unusual pictures. Thus, only those people who had visited the Hyperscore stations at the open house prior to the concert would have even seen the Hyperscore interface, and even for me, very used to the Hyperscore graphics, it was difficult to follow along with the music.

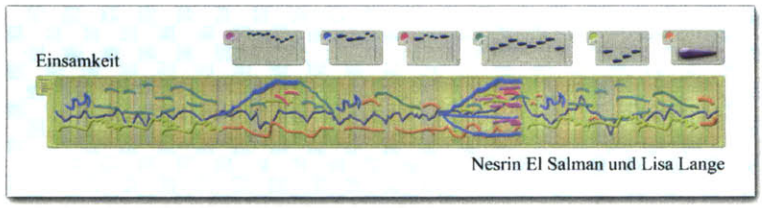
Thus, it's likely very few concertgoers actually made the connection between the music they heard and the graphics they had been given. We hope that the concertgoers did, in fact, understand the "basics", that the pieces had been composed by children using an MIT software package, and the graphics represented the notation system the children had used.

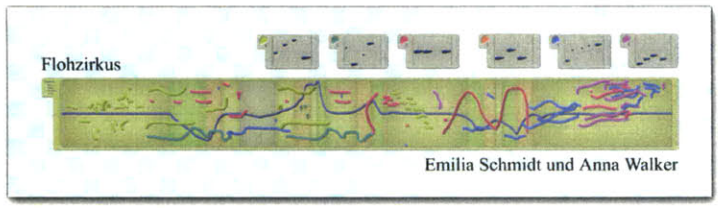
5.a.h The Pieces Produced in Berlin

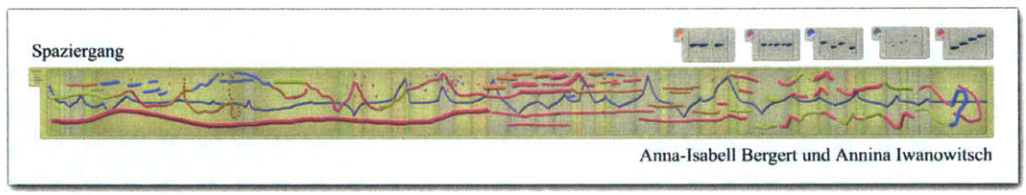


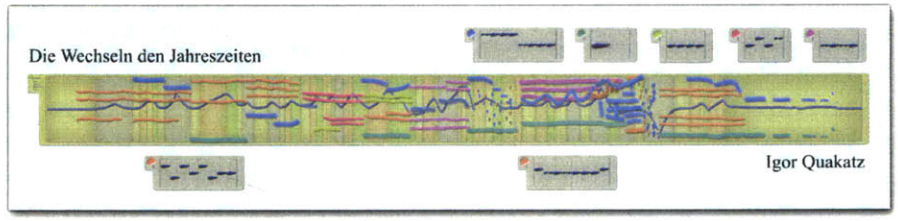












5.b Dublin

- **Concert and Open House**

  Date: Sunday, April 7, 2002
  Location: The Dublin Royal Concert Hall
  Orchestra: The National Symphony Orchestra of Ireland

- **Hypersccore Workshops**

  Date: Monday, April 1, 2002 thru Friday, April 5, 2002
  Location: The ARK Children's Center in Dublin
  Children: Children enrolled at the ARK

After the Berlin workshops that had, in essence, taught us the basics of teaching children to work with Hyperscore, repeating the same thing in Dublin two months later was considerably simpler. A major difference was that the children spoke English, thus no translators were needed and we could get to know the children on a much more direct level.

The computers used in the Berlin workshops had been shipped to Dublin and set up at the ARK Children's center, a place in lower Dublin that served as an arts-and-crafts activities center. Many of the children who regularly frequented the ARK were interested in Hyperscore (as well as the other Toy Symphony instruments), thus there was no need to search for a sponsor school.

When we arrived the day before the workshop week, we discovered that many of the children had been working with Hyperscore for several days. But unlike the random fragmentary content the Berlin children had prepared before their workshops, these children had been working with Kevin Jennings towards the final goal, and they all had fairly complex pieces in production.

5.b.a Fiachra, Rachel, and the Rest

There were two children with notably unique works. The first, Fiarchra, was at 14 the oldest child we ever worked with. (All others in all three cities were in the 9-12 range.) Fiachra appeared to be extensively experienced

and did not use the harmony modes. Rather he turned harmonization off and drew mostly short straight strokes that played his motives without modifications. Fiachra made over a dozen motives, carefully positioning each note's pitch by hand. While to my ears it sounded repetitive, it was dramatically different than the rest of the pieces.

The other unique child was Rachel, who was clearly the most visual of the children. Her pieces had lots of spirals and wavy lines and clusters of droplets that had clearly been arranged, on the Hyperscore canvas, for their visual effect. However, unlike the unfocused children we'd seen at our test workshops in Cambridge who had drawn happy faces and never cared about the sound, Rachel was clearly focusing as much concentration on the sound of each mark as its look. She simply treated both as equally a part of her finished work. To her, the Hyperscore interface was not merely an irrelevant *presentation* of the final audio product, but was part of the art itself.

(Please see Rachel and Fiachra's pieces in section 5.c.d, below.)

Overall, the children in Dublin appeared more musically trained than our Berlin children had been. Certainly several had taken music classes before, and most of their pieces followed recognizable patterns, ABA or ABAC for example. However, while their pieces showed structure that had been lacking in Berlin, there seemed to be less variety, (with the exception of the Fiachra and Rachel pieces.) It was suggested that what formal teaching they had experienced had limited these children's imagination, and their pieces tended to sound the same. They were all nice, warm, harmonized textures that built to climaxes in predictable waves.

5.b.b Steady Progress

Back in Berlin, we had needed to focus every day on a different task: motives, textures, contrasts, melodies; and this had involved coding new features or composing new pieces each night for inclusion into the following day's lesson. We were cautious at every step, being our first experience, and we met after each lesson to discuss the progress and plan our next day. Every day we learned something new about how to run a workshop and thus the Berlin week sticks in my mind as a progression of five very different experiences each building on the one before.

However, in Dublin the children seemed to be fine on their own. The unique features of Fiachra's and Rachel's work, and the good quality of the rest of them, were all things noticed on the first day, Monday. In the following four days all the children tweaked their pieces, managed to make them longer

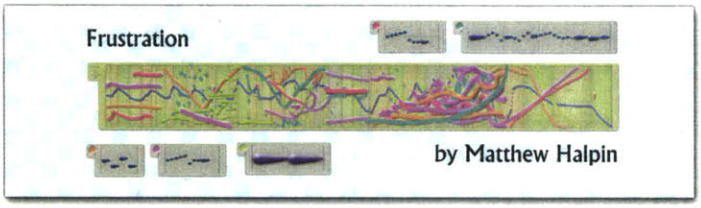
before we decided that they were too long and had them make them shorter again, but more than half of the work completed on Friday has existed on Monday. By the third day we knew which pieces we were going to pick and many children didn't show up on the last two days; considerably more time remained to transcribe the selected pieces.

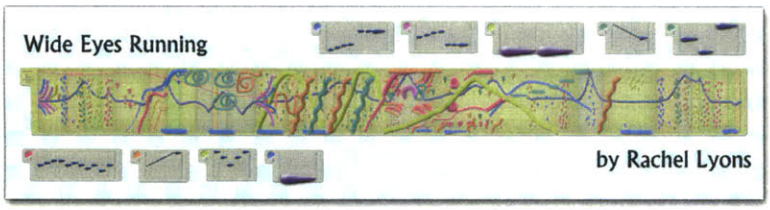
5.b.c The concert itself

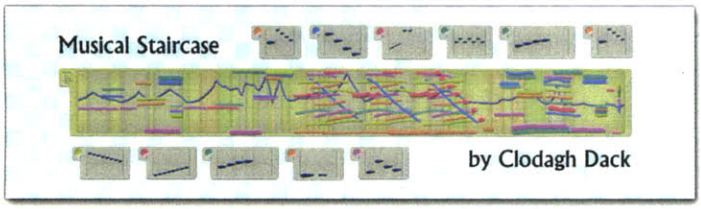
The transcription and rehearsal progressed as before; unfortunately, despite considerable discussion with the Royal Concert Hall, it was again not possible to project a moving image of the Hyperscore graphics during the concert. However, in addition to the graphic posters and handouts that were presented to the concertgoers (which had been prepared in Berlin as well), a large 20-foot-wide glossy banner was made of Clodagh Dack's piece, unfurled on stage.

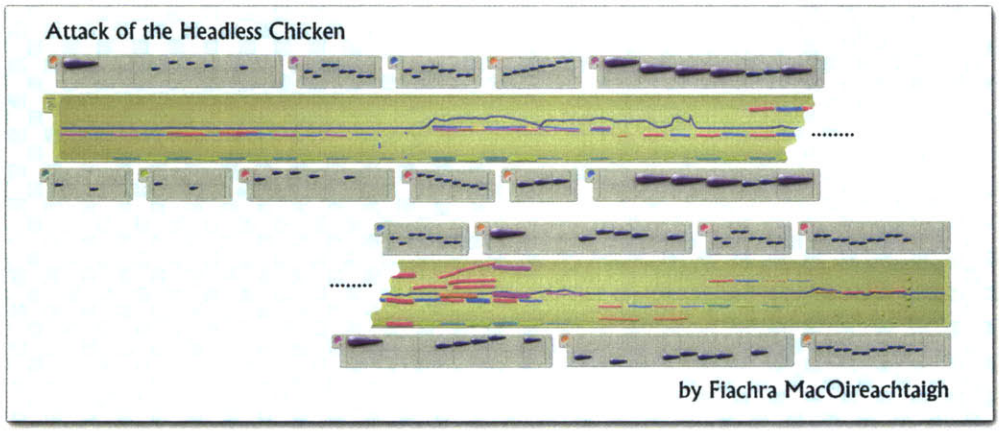
Tod Machover, doing the announcing for the Dublin concert, described the program in detail using the banner as a visual aid, pointing out different marks and explaining how the interface worked. Many audience members reported understanding the correspondence between the music and the graphics they had been given, suggesting that the on-stage explanation was an excellent idea.

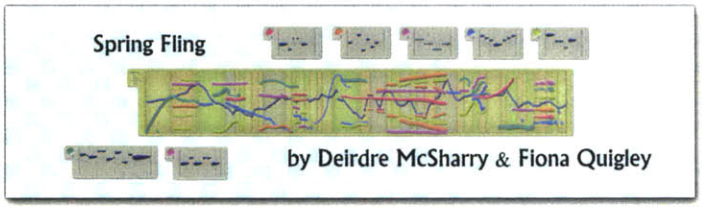
5.b.d The Pieces Produced in Dublin

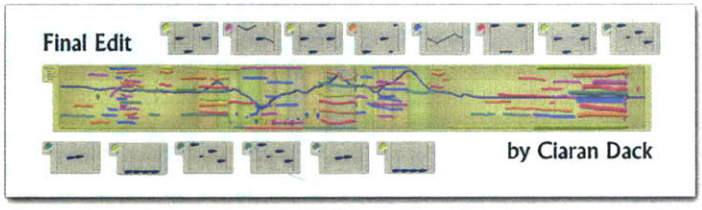












5.c Glasgow

- **Concert and Open House**

  Date: Sunday, June 2, 2002
  Location: Royal Concert Hall, Glasgow, Scotland
  Orchestra: BBC Scottish Symphony Orchestra

- **Hypersccore Workshops**

  Date: Monday, May 27, 2002 thru Friday, May 31, 2002
  Location: Barrowfield Primary School, Bridgeton
  Children: Barrowfield Primary students

The final Toy Symphony concert in Glasgow was handled largely through the BBC, which arranged for us to work with a group of students at Barrowfield primary school in Bridgeton. Barrowfield is a small school located in a reasonably low income area outside Glasgow, and the children enrolled there had the least amount of musical background of all the children we'd seen so far. They also had access to the least amount of personal support and encouragement, yet they still succeeded in making pieces of which they were proud.

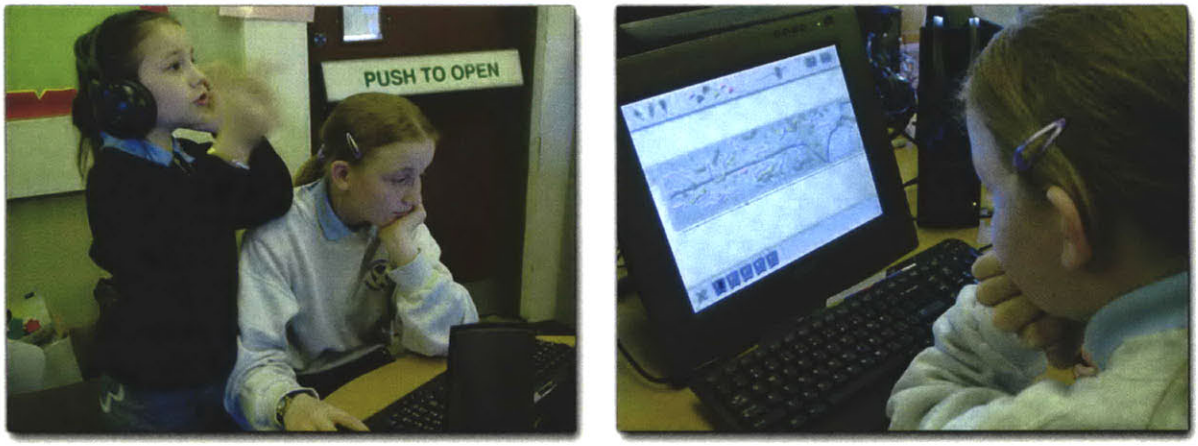


Figure 42: Dionne Coutts and Jasmin Riggins build Hyperscore pieces at the Barrowfield Primary School, Glasgow Workshops, Monday, May 27, 2002.

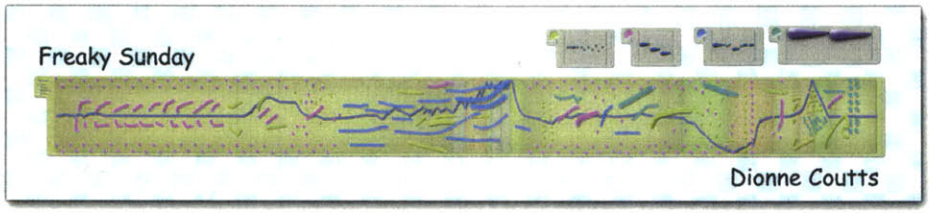
From this perspective the workshop in Glasgow was perhaps the greatest success for Hyperscore.

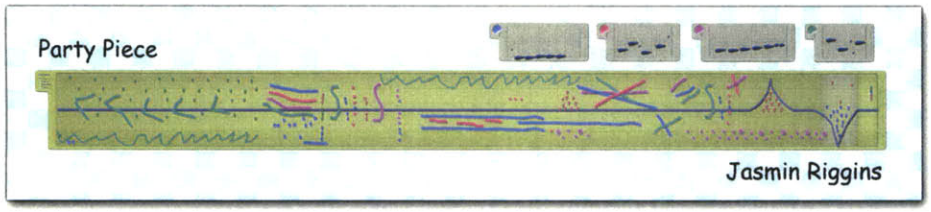
5.c.a Projection

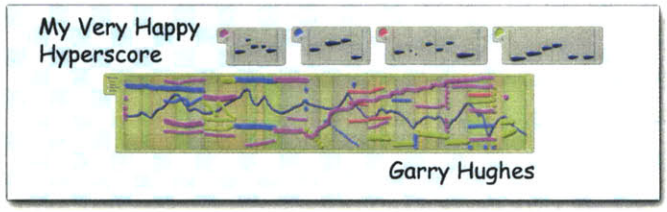
Unlike our two previous venues, the Glasgow Royal Concert Hall was able to set up a projection system, as we had wanted in the previous two concerts. This finally permitted an audience to listen to the children's work while relating the sound to its visual appearance. All of the feedback, from parents, children, and other audience members, concerning the projection was enthusiastic.

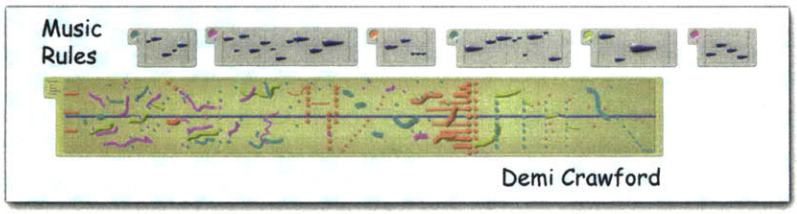
Figure 43: The orchestra plays transcribed versions of pieces composed by children using Hyperscore; while the original Hyperscore representation scrolls by in the background. Glasgow Concert, Royal Concert Hall, Sunday, June 2, 2002.

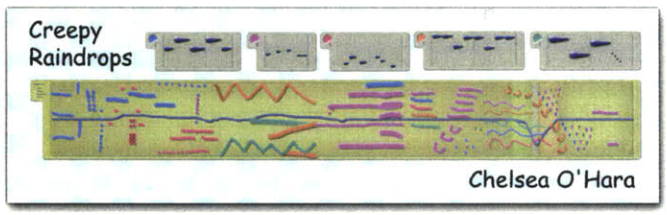
5.c.b The Pieces Produced in Glasgow

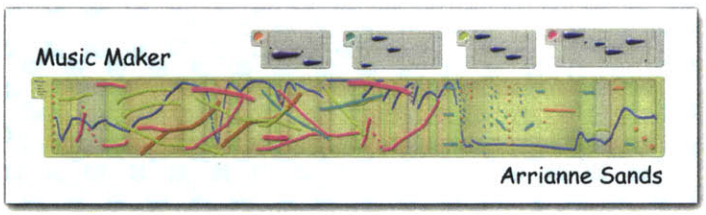












Chapter 6: Evaluation & Future Work

This chapter summarizes both the positive and negative reactions we have received from people who've tried the program, and what the experience teaches about interface design. Ideas for the future are listed, both near term ideas that are actually in the process of being implemented, and blue sky suggestions that look further out.

6.a Positive Aspects

During the Toy Symphony concerts a large number of people from the general public were allowed to examine and use Hyperscore directly, and the feedback received was overwhelmingly positive. Teachers, children, reporters, and interested adults have all declared the program exciting and fascinating, and official press reports on Hyperscore have been positive. [Times, 2002] Thus we believe we have succeeded in the most important way: we've made people realize that they can make music, who would not have realized it otherwise.

6.a.a Hyperscore is at a good middle ground between user control and computer help.

One of the greatest issues in the design of Hyperscore was the degree of computer assistance offered to the user. At one extreme, a computer-aided composition program could accept the user's input completely literally, playing any mark the user made without regards to its musicality. Such a program would serve merely as a repository of notes, and would not be providing intelligence.

At the other extreme, the program could treat the user's marks as merely suggestions, or parameters, and ensure that it produced good music at all times. Such a program could be presented as using "intelligence" to clean up inputs marks and make them musical with no regard for the experience of the user.

Both of these extremes have flaws. In the former, although the user enjoys complete control over the notes played, he is likely to discover that it is hard to make good music without assistance; the user may experience frustration, eventually boredom. At the other extreme, the user may be initially

delighted by the low degree of effort required to produce a nice result, but might feel a lack of control.

The current version straddles a middle-ground between these two extremes. The user's marks are played faithfully with only the small changes required by the harmonization algorithm, which simply takes the notes, as drawn by the user's contours, and adjusts each note in pitch to the closest pitch in the predetermined chord sequence (controlled by the harmony line).

6.a.b Users can create a great deal of interesting material quickly

Here, speed is important. Programs that require each note to be placed one by one require a large number of clicks for each additional second of composition length, and risk being perceived as too time consuming. Further, a great deal of thought is required for each note; if the user intends his work to sound good, he must place each note in a specific place with respect to the notes around it -- this may further slow down the experience.

We have encouraged children to approach Hyperscore composition with a try-it-and-see attitude. The user is to make a mark, listen to its effect on his composition, and then modify the mark until the effect is considered pleasant. There is some question as to whether this approach is appropriate for teaching serious composers, but there is no question that, so long as the child is not bored, the approach will work. If the program required the user to place each note one by one, we could still encourage the try-it-and-see approach, except here, the child would consider each note separately. It would just be very slow and the child would likely get bored before composing very much.

The ability to duplicate motives and entire blocks of notes can overcome the speed issue, as in Jeanne Bamberger's program. Once the user has created a block of notes in *her* program, he can easily create very long patterns using these blocks as components. However, one pattern repeating over and over again rapidly becomes boring as well. A piece must have variety in order to keep one's interest.

In Hyperscore, the user can repeat a motive by drawing a long stroke in the color corresponding to the motive. However, boredom is averted by providing two different methods by which the user can introduce variety quickly. First, the user can adjust the stroke line vertically, thereby transposing the motive up or down in pitch or altering its contour. Second, the user can adjust the harmony line, altering the chords and key used in the piece. That both of

these operations are implemented as a single click-and-drag means that the child can produce seemingly unlimited variety with very few strokes, in very short time.

In summary, limited to note placement, the child would be able to make only a very short amount of interesting music before becoming bored. Allowed to duplicate blocks of notes, the child would be able to make a large amount of repetitive music, and would become bored in that manner. Hyperscore's line-drawing-interface solves both of these issues, permitting the child to very quickly create a large amount of interesting music. Thereby the child is pleased, does not get bored, and instead is able to channel his excitement toward making the music interesting *in just the way he wants*. And there, we have composition.

6.a.c Reshaping strokes

During the testing, every single person who saw the dynamic curve editing function operate reacted with delight. The ability to modify a curve by grabbing and pulling seemed much simpler than the chore of having to manipulate tiny control points, despite the fact that the latter would technically offer the user more precise control.

6.a.d An attractive interface attracts curiosity

That the Hyperscore interface does not look like other Windows programs appears to be a point in its favor. One of the results of this non-conventionality is that people eager to try something new feel motivated to download and experiment with the tool, where they would be less motivated by a program that appeared just ordinary.

This may be quickly dismissed as a trivial human factor, but it appears to be actually an important one. In an age where every idea has dozens of competitors, any component that sets one above the rest is an important one. Hyperscore's interface has an unusually high "gee-whiz" factor which dramatically increases the degree of interest from those who examine it.

6.b Negative points

6.b.a Problems adjusting the harmony line

While the user is allowed to control the shape of the harmony line easily, the presence of the red/green/gray boundaries denoting the sharp boundaries dividing harmonic sections has the effect of causing the user to be greatly concerned about positioning them precisely. Frequently users find themselves tweaking the harmony line up or down by tiny amounts over and over again in an attempt to cause the computer's interpretation of section boundaries to land in a target spot.

This suggests that a future version of Hyperscore should include only the boundaries, and offer the ability to adjust these colored segments directly, omitting the harmony line entirely.

6.b.b Usefulness of zooming interface

One of the greatest problems with the current version is that it is difficult to justify the panning and zooming interface given the current fact that each composition in Hyperscore consists of only a single, central yellow window (ignoring the small assortment of surrounding motive windows).

If each composition is always going to be a single yellow window, it seems logical that displaying the yellow window in a fixed position on the screen, and supplying a scrollbar, would be easier than all this zooming and panning. You would lose the ability to see the whole composition in miniature, but gain in familiarity in that the program would look more like other programs. Indeed, if one thinks of a composition as "just one yellow window", then the zoomable view is really not justified. There is no reason to be able to pan the background surface if the surface will always be blank except for one yellow rectangle in the middle.

Rather, the zoomable user interface begins to shine only in a situation where a composition is a two dimensional arrangement of *many* different components -- multiple yellow windows connected by a higher-order timeline curve, multiple yellow windows playing at the same time, multiple objects connected in intricate ways. If a composition really were such a "two

dimensional machine", then the capability to examine and edit this machine through any viewport would seem essential and useful.

And that is, at the moment, the clear strength of the Hyperscore zoomable-canvas as it stands: *That we can look at this canvas and see, yes, the program can be extended this way.* Right now, when you hit "play", we have one yellow window playing. But next month, perhaps, we could have two playing together, each with different harmonizations turned on, or each with different instruments going (once we move beyond just string sounds). We can imagine other blocks that apply transformative algorithms on the notes, or audio effects on the sound, and imagine, visually, how such blocks would be connected to the others upon the canvas. The Hyperscore canvas is a space we can play with. A space to populate with more capabilities in the future.

If Hyperscore did not have the zoomable-canvas; if the zoomable canvas had never been thought of, and instead the display had merely extended the "Hyco" interface, showing only the contents of one yellow window which could be scrolled from side to side -- if Hyperscore was like that today then all these clear ideas for how to extend Hyperscore would not be clear at all. The screen is full, we'd say. Where do we put more strokes or more capabilities? "Perhaps we should add more tab-controls or dialog boxes", we'd think, though doing so would make the GUI more complex and less accessible to children. And that way would be the path of the complex interface.

Thus the zoomable canvas is the correct decision for Hyperscore because of its extensibility, and the things planned to be added to it in the future. But the fact remains that (in my opinion, at least) its existing features do not by themselves provide a compelling justification for the ZUI paradigm.

6.c Future improvements

The Hyperscore interface can be extended and improved in many ways. Many of these ideas are under implementation or likely to be completed in the next few months, while other possibilities represent further ideas that this experience has generated, but may never actually be implemented by this team.

6.c.a Advancing tonal composition

There are a number of ways in which the current program could perform its current function better.

• The computer should provide assistance regarding timing. When positioning a stroke, it should "snap" to line up the starts of strokes, or the starts of strokes to the ends of others. As the user drags a stroke, whenever the horizontal position of the stroke nears lining up with important positions, like the starts or ends of other strokes, it should jump into place. Currently, the program provides no feedback when such good alignments are possible.

• The ability to invert or retrograde motives or strokes should be provided. It should be possible to see, graphically, the duration of a stroke compared to the duration of its motive. How many times the motive will loop on a given stroke should be visible, with marks on the stroke corresponding to the start of each repetition. Other strokes should snap to these as well.

• It should be possible to set up multiple Stroke Windows to play simultaneously. Stroke window playback should be triggered by graphical toggles attached to other Stroke Windows, enabling the user to assemble a composition consisting of a network of multiple Stroke Windows. Currently, a composition can only consist of a single Stroke Window.

• New timbres and new instruments should be supported, including drum and horn sounds, pianos and guitars. Currently, a single binary timbre selection is provided by a texture on each stroke. Instead, the timbre could be associated with the Stroke Window. In this case, all strokes within a single Stroke Window would have the same timbre, and the user would arrange multiple Stroke Windows to play concurrently in order to enjoy multiple kinds of instruments playing at once.

• The Motive Windows and Stroke Windows could be unified, both replaced by a single type of "Score Window" that would hold both free notes, and pen strokes whose effect would be to insert morphed copies of whatever notes were bound to that pen. The contents of any "Score Window", in turn, could be mapped to a color to permit the user to create other "Score Windows" using the content as a component. This formulation would permit hierarchies to be built with an arbitrary number of level instead of the two allowed currently.

In such a formulation, harmonization would be removed from the stroke windows, and be implemented as a separate window which could be docked to any "Score Window".

6.c.b The Simulation Surface

A completely different direction of improving Hyperscore would be to consider other kinds of "sound objects" that could be created on the canvas. Currently, Hyperscore support only the two "objects", specifically Stroke Windows and Motive Windows, the purpose of which is to sequence tonal notes. However there are numerous other methods that could be used to produce sounds.

- For example, the canvas could support objects which behave autonomously and make sounds as they interact with each other. Such objects could react to static features placed on the canvas, such as sources that would attract or repel them, or force fields to move them around in paths that vary in time.

- The canvas could support physically simulated sound sources such as wooden blocks or resonating cavities. For example, the user could draw a shape using a brush representing "wood" or "metal", then fire a projectile at it and hear its sound. The user could add a cavity into the shape, fire another projective, and hear the effect of the cavity upon its tone. Thus the user could, in effect, "paint" his own unique instrument.

6.c.c Hyperscore as an extensible toolkit

Finally, a concurrent goal of Hyperscore is to create an environment that can be arbitrarily extended by other programmers to create different sound programs with new gesturally-controlled algorithms, effects, and sound sources.

By reworking some of the internal architecture, Hyperscore could be restructured so that the various objects explored in the previous section could be independently designed for a common canvas. In this formulation, the "Motive Window" and "Stroke Window" of the current Hyperscore, or the "Musical Creature" or "User-painted instrument" objects suggested above, would all be plug-ins loaded at runtime into the zoomable framework that, by itself, provided only a canvas and the mechanism for its plugins to communicate.

Such a structure would allow a formal API to be published which would permit any interested programmer to invent and implement new objects or applications using the intuitive framework.

Chapter 7: Conclusion

The End so far. This thesis has presented a novel interface for a music composition program for the musically untrained. The program employed a zooming, direct-manipulation interface to good effect, and has shown line-drawing to be a powerful input method for composition.

The program has been extensively battle-tested in three separate public concerts, where dozens of children engaged the program for weeks to produce string orchestra pieces. A total of twenty pieces, all excellent 2-minute long compositions, have been produced so far, and more concerts are scheduled.

The program has been made available for download and so far hundreds of people have downloaded it and many even got it working. A bulletin board was designed to encourage a community to form around Hyperscore and this appears to be progressing.

We conclude that a zooming direct-manipulation-interface using freely drawn lines as the main form of input is an excellent way for musically untrained users to compose music, and become introduced to music.

Appendix A: User Guide

This appendix presents instructions in a tutorial-form on how to download and use the program. This tutorial is written to be read by a musically-untrained novice user who might have Hyperscore open on his screen, following along with the instructions.

Hyperscore will run an a PC running Windows 95, 98, 2000, ME, or XP. Windows NT or 3.1 are not supported, and neither are Macintosh or Linux computers, however we are considering porting Hyperscore to these latter platforms in the future. The program will run on computers as slow as 200 MHz, requiring a minimum of memory and less than 10M hard disk space. In addition to a keyboard, mouse, and display, your computer must have a sound card capable of playing MIDI notes, and a graphics card with 3D graphics acceleration.

However, if you have these requirements you may install Hyperscore by downloading an installer from our website. The best way to use this tutorial is to use Hyperscore and follow along as we describe, step by step, how to make a short, simple, but pleasant piece.

A.a Downloading

To download the installer go to the Toy Symphony website at http://www.toysymphony.net and browse for Hyperscore. You should be able to find the "Hyperscore Showcase" where the download link is prominently displayed. Download and run the self-installing executable, and it will place Hyperscore and its datafiles into a folder on your C drive, and place an icon to the program on your desktop.

Finally, before you may run Hyperscore, you must also make sure you have DirectX8 installed. DirectX is Microsoft software that provides an interface for software to communicate with graphics cards. The Windows XP operating system comes with this installed by default, but if you have an earlier version of Windows you'll need to install it manually. The installer is available from our download site, or from Microsoft directly.

With DirectX8 and Hyperscore installed, click the Hyperscore icon to begin the program.

A.b Starting Hyperscore

When Hyperscore appears you are presented with a blank blue-and-white checkerboard canvas on which you will make your composition. The following figure shows labels for the basic tools in the interface.

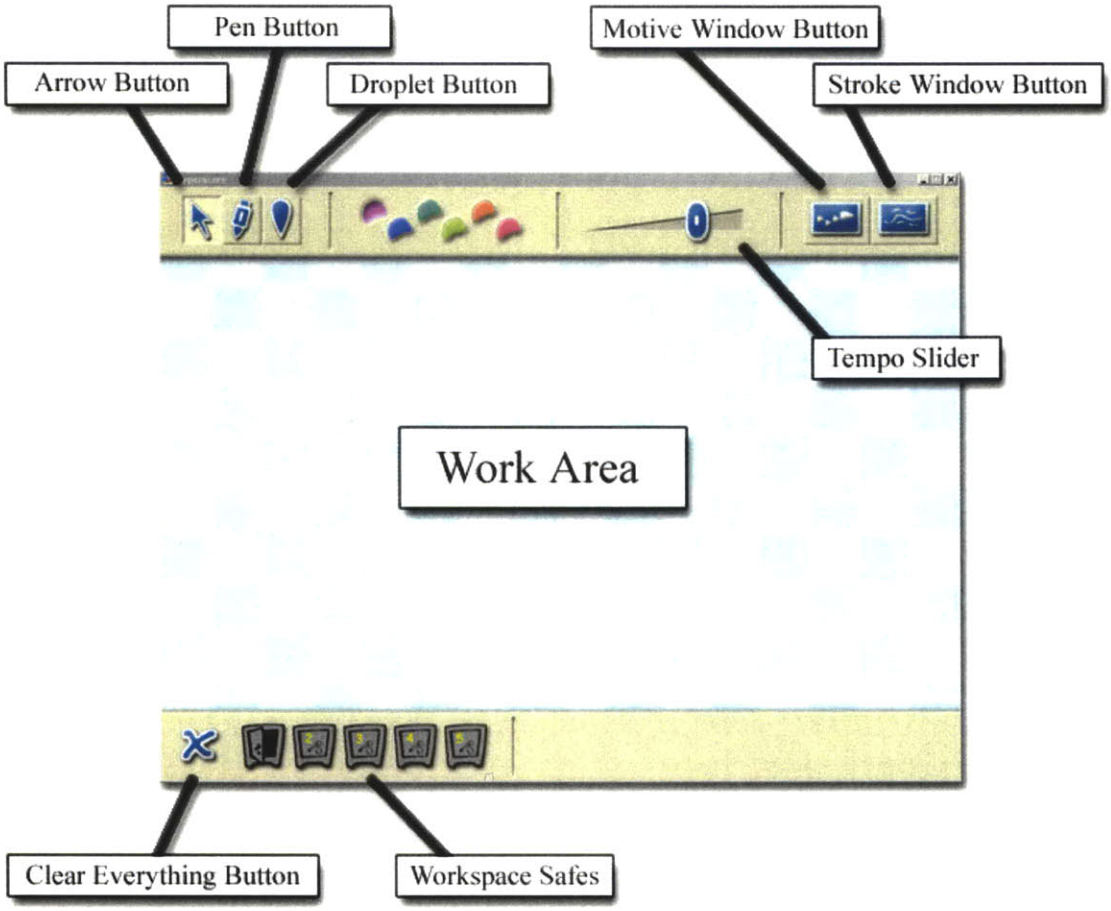


Figure 44: A screenshot of Hyperscore whose workspace is blank. All the buttons are labelled.

You build your music by creating musical objects on this canvas. There are two types of musical objects, Motive windows and Stroke windows, and we will quickly explore both in this tutorial to give the user an overview of how the program works. The Reference Guide, to follow, will carefully describe each feature in a top down order.

A.c Creating a Motive Window

First you will create a Motive Window. Click on the *Motive Window Button*, and then click anywhere on the checkerboard canvas. A motive window will appear.

Figure 45: Two clicks are used to create a blank motive window. Notice the window has lines etched into it, defining a time-versus-frequency grid.

In this window, you can create a short, single-voiced motive. Click on the *Droplet Button*, and then click a few times on the Motive Window. You will be able to place notes anywhere you want to on the Motive Window, but you can only have one note playing at any given time. Press the space bar to play the motive.

Figure 46: After clicking on the Droplet Button, you can add notes to make a motive.

Move the notes around until you like your motive. Later we'll learn how to edit motives in many ways, including adding notes that are longer or shorter, or making the motive longer or shorter. But first, we'll see how a motive gets used in the other kind of musical object, the Stroke Window.

Notice the pink color blotch on the side of the Motive Window. This means that your motive has been associated with the color pink. In a moment, you'll create a stroke window and layer many copies of these motives together.

A.d Panning and Zooming the Canvas

Before you add more content, you should become comfortable with moving around the canvas. You are not limited to the screen real-estate presented when the program first starts up, Hyperscore provides a vast canvas that stretches many screen widths. You can pan your canvas around by holding down the control key, then clicking and dragging with the mouse.

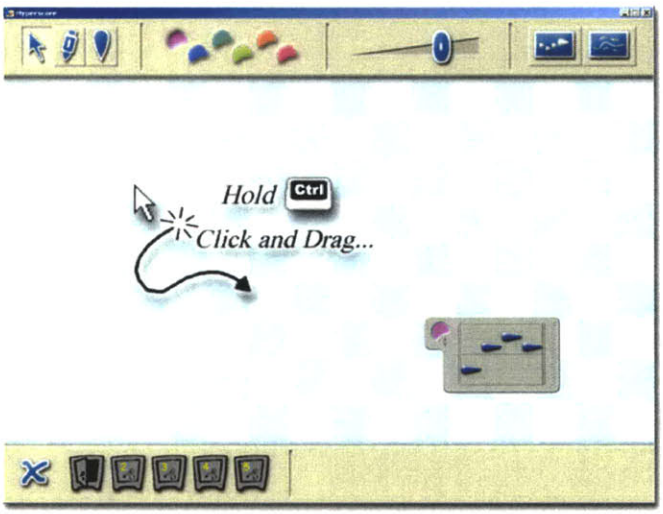


Figure 47: You can pan the canvas around to bring more space into view.

You can also zoom the canvas in or out by pressing one of the F1,F2,F3, or F4 keys. Each of these "F" keys takes your view to a different zoom level. In addition to changing zoom, these keys also pan the canvas to center the location of the mouse. Thus, when zooming in, you should remember to put the mouse over item you'd like to see more closely.

Hyperscore starts at the F2 zoom level, a level which permits several motives and a good section of a stroke window to be on screen simultaneously, while still displaying all at a resolution sufficient for editing. Precise editing is easier in F1 zoom (shown in the figure), while the F3 and F4 levels permit a large composition to be scanned as a whole.

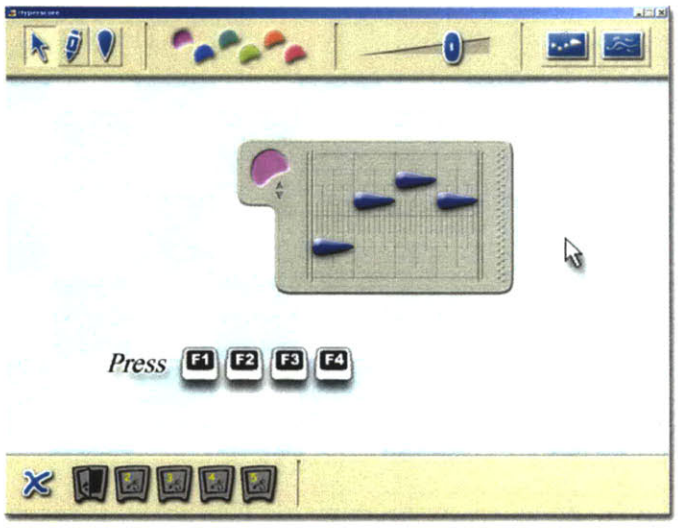


Figure 48: The first four function keys switch between four levels of zoom. You can zoom in to see your motive more closely.

Practice moving your canvas around until you are pretty comfortable with this. Once you're making longer and longer pieces, you'll need to use these keys to move around many screenfulls of your work.

A.e Creating a Stroke Window

Next, you can create a Stroke Window. Make some space to fit a Click on the *Stroke Window Button*, and then click anywhere on the checkerboard canvas. A stroke window will appear. Try to place it next to the Motive Window above.

Figure 49: Two clicks are used to create a blank stroke window. Strokes drawn in this window will layer together to make a piece.

On this window, we draw with the pen. Click on the *Pen Button*, and make sure the pink color is selected in the color swatch -- the color corresponding to the motive. Draw a short straight stroke on the yellow window, as shown. Hit the space bar to play!

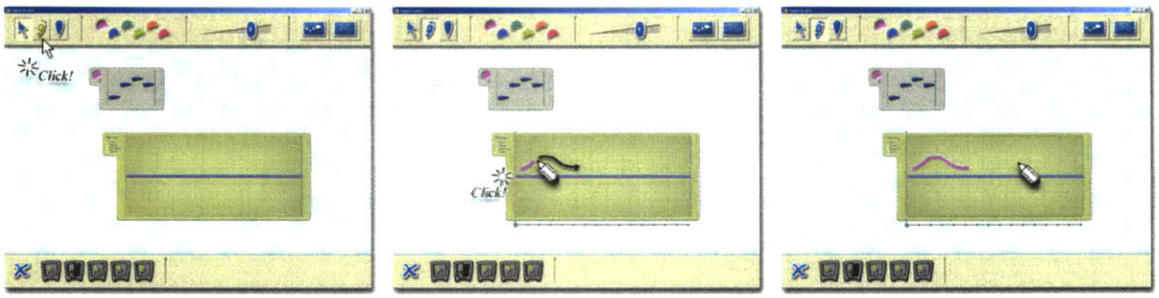


Figure 50: Click on the Pen Button, and thendraw a stroke in the color matching the motive. As the line plays, the associated motive will loop.

You can move this line up or down with the mouse to change the sound. Moving it lower, lowers the pitch and lifting it higher raises it. You can also reshape the stroke by dragging the stroke with the right mouse button. When you draw a stroke which is curved, up or down, then the motive's contour will change to follow the bends in your stroke. By carefully shaping

different strokes, you can get many different patterns out of a single motive! Notice that each stroke starts playing at the time corresponding to its leftmost endpoint, and stops playing at the time corresponding to its right, so there's no point in drawing strokes like circles, vertical lines, or self-intersecting marks. It makes the most sense for strokes to go from left to right with vertical variations.

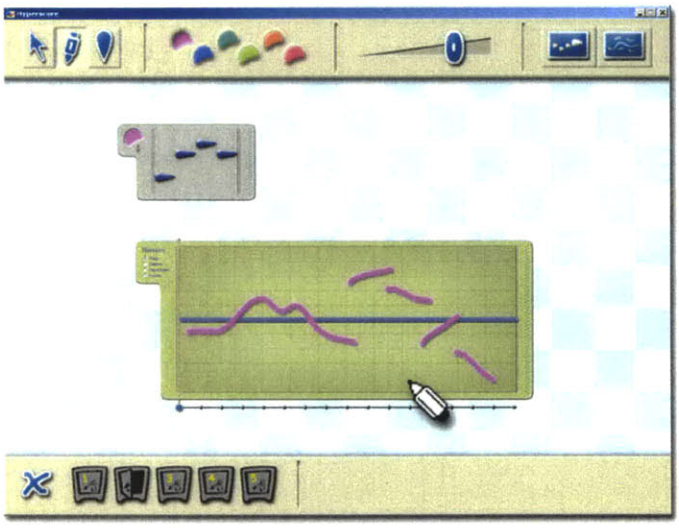


Figure 51: Differently shaped strokes play different

Try these things one at a time, and play the piece after each to see how changing the stroke changes the sound that you hear. Experiment by drawing many lines. Finally, you could also try making another motive and blending different motives together.

A.f Timbre and Volume Control

When you want to draw new strokes you must have the *Pen Button* selected. However, when you have the *Arrow Button* selected, you can change many things about curves. To try this, select the *Arrow Button*, then click on a stroke that you drew earlier or drag a rectangle around a group of notes. The notes appear with a selection rectangle around them.

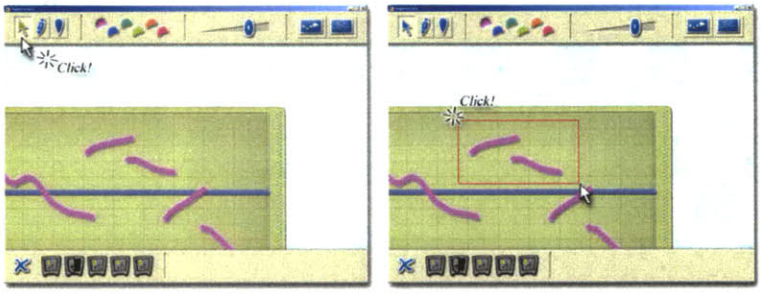


Figure 52: Click on the Arrow Button and drag a rectangle around some strokes to select them.

At the upper right corner of the rectangle several buttons appear which allow you to effect several properties of the selected strokes.

Click on the bowed/plucked timbre button and observe how the stroke changes appearance, then play the piece and listen to the change in sound. Next, *click and drag upward* on the volume icon and see how the stroke becomes thicker or thinner, and listen to the change in volume. Listen to the softest and the loudest you can make a motive.

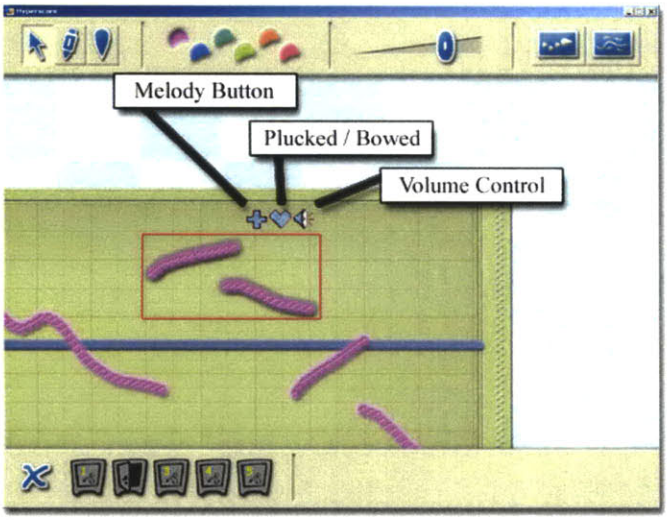


Figure 53: Selection rectangles come with several controls.

The final button, the "Melody Button", is relevant only if harmonization is turned on. Harmonization is discussed in the next section, so for now, leave the "Melody Button" untouched. Later, come back and listen to how a stroke harmonizes differently when the "Melody" flag is engaged.

A.g Harmony

Now that you know how to draw strokes, bend them, and affect their properties, it's time to look at harmonization, which is a way Hyperscore can change your notes to make them sound better together. To test this, select the *Pen Button* again and make a lot of strokes, some flat, some curving up or down. Make sure you have several strokes playing most of the time.

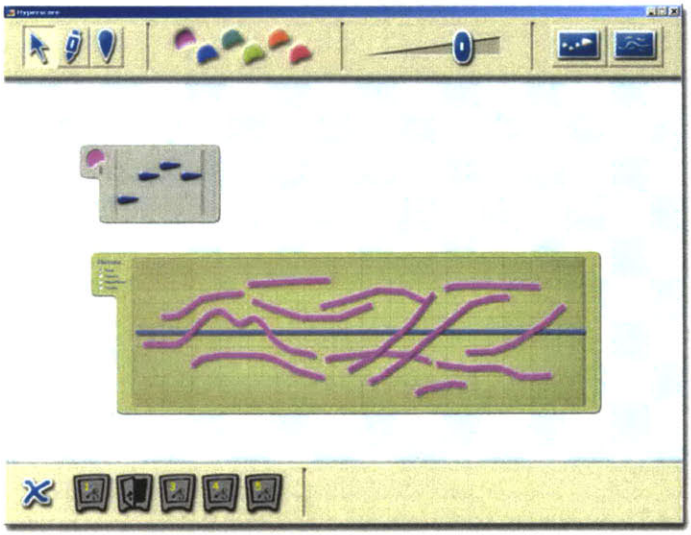


Figure 54: To test harmonization, draw many strokes so several voices play at once throughout the piece.

When you make music with Hyperscore, you tend to get better music if you go slower than this, and add strokes one by one. Every time you draw a stroke, you

should play the music again and listen to hear what you've added, and adjust your marks frequently to make sure it sounds how you want. But for now, just draw a lot of strokes all at once so we can learn about harmonization.

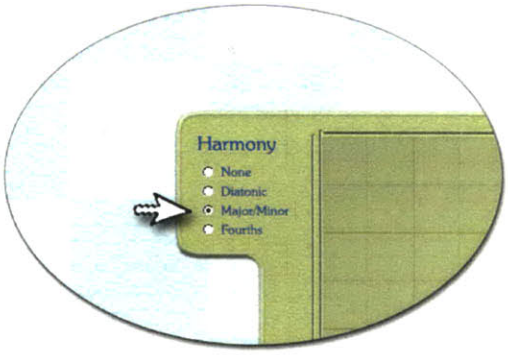


Figure 55: Switch the radiobutton to the third entry to enable harmonization.

Once you have all your strokes drawn, press play and listen. It probably sounds like a real mess, with lots of notes playing that don't sound good together. This is because, by default, Harmonization is turned off. It's controlled by those little toggle buttons on the side of the Stroke Window. Switch the Harmonization to "Major/Minor", the third option, and listen to it again.

Does it sound better? Is it more or less interesting? With harmonization turned on, Hyperscore will make sure most of your notes follow a sequence of chords -- and you can control the chords that it uses using that central dark line that we haven't talked about yet.

When you grab and move the central dark line, called the "Harmony Line", colored regions appear in the background. To begin with, lets listen carefully to the effect of a simple change in the harmony line -- so play with the harmony line until you get a single red and green area. Listen to how the notes sound in this region.

Another thing you can do is create a *gray* region. The gray regions come where you draw a spiky shape. To make one, take a flat part of the harmony line (click to grab the harmony line and move your cursor quickly right and left to flatten a part of the harmony line), then grab a single point of the harmony line and slowly drag it upward to make a spike.

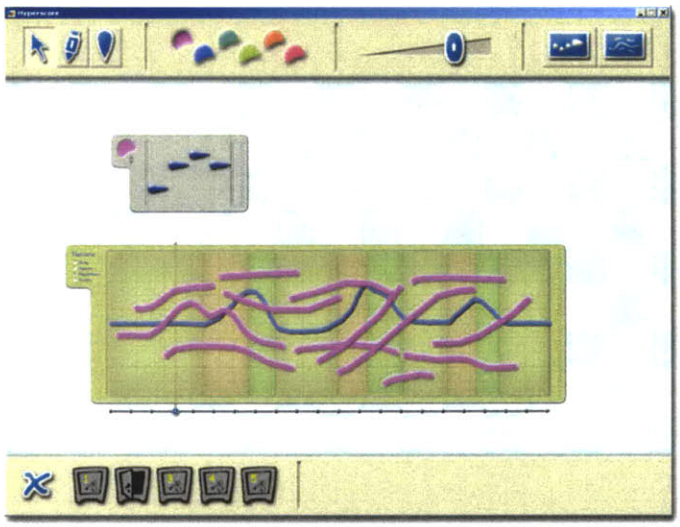


Figure 56: Shaping the central line into little bubbles produces little red-green regions. These regions will possess chord variations.

The gray regions change the

music's key. This means that the music changes when it passes through the gray region, and then remains changed, in contrast to the red and green regions we tried before. The red and green regions have an effect only where they are, while the grey region affects everything forward of itself until the next grey region.

The taller you make your spike, the greater the extent to which the music changes. Experiment with lots of different colorful regions in your harmony line to add variety to your piece.

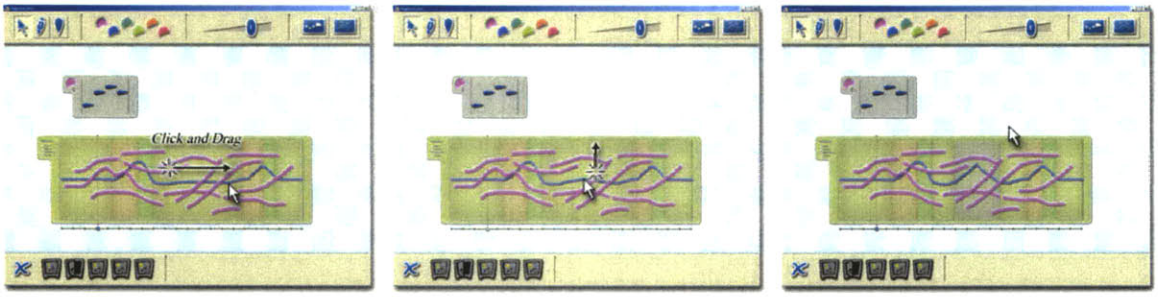


Figure 57: Create a spike by flattening a region of the harmony line with a quick horizontal stroke, then slowly raise a point in the center. The resulting gray region effects a key change.

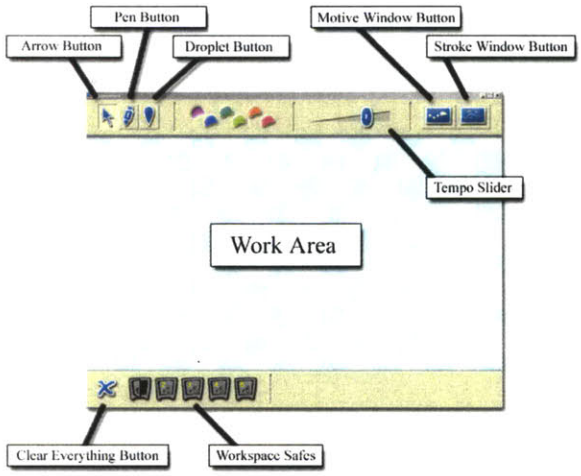
A.h Explore and Play

Now, you're ready to try making something more complicated. Try adding different motives, try building careful structures of a few motives playing against each other. Duplicate sections to make your piece longer, but add variation at each stage. Remember the trick to composition is to make sure that your listener never gets bored.

Good luck!

Appendix B: Reference Manual

The preceding "tutorial" introduced many of the capabilities of Hyperscore in an order and style appropriate for a person sitting down at the interface for the first time, who wants to get creating content as fast as possible. The reference guide to follow presents the capabilities of Hyperscore in an order that matches the program's internal organization.



To begin, examine again the Hyperscore interface. Notice that of the five rectangular radiobuttons at the top of the screen -- *Arrow Button*, *Pen Button*, *Droplet Button*, *Motive Window Button*, and *Stroke Window Button* -- One of these buttons is always selected. Therefore, this defines a basic program state the user can directly control -- the program is either in *Pen Button* mode or *Arrow Button* mode, etc.

The program's behavior depends on this state. The user can do things with the mouse and keys when *Pen Button* is selected that he cannot do when *Arrow Button* is selected, and vice versa; the user should remain aware of which button is selected as he uses the program.

B.a Focus

During the use of Hyperscore, the program's **focus** is a second basic state. The program's **focus** is always in one of three situations: either a Motive Window has **focus**, or a Stroke Window has **focus**, or nothing has **focus**. You can tell that a window has focus because the play button and its associated graphics appears under a window with focus. Since the operation of the various buttons and keyboard shortcuts all depend on which of these three focus-states the program is in, it is reasonably important for a user to be aware of what has the focus while using Hyperscore.

(This is an aspect of the user interface that can cause confusion, and should be improved. I believe that the program should make it a lot clearer which window has focus. At the very least, the focused window should appear

brighter, or posses a dark border. Even better, the entire workspace could fade to a low-contrast grayscale, with the exception of the focused window remaining in bright color, in order to make it visually obvious to the user that he is in a state where it is expected he will manipulate the contents of that one window only.)

B.b When Nothing has Focus

Naturally, most of the commands and operations possible in Hyperscore are operations on a particular window, whether it's a Motive Window or a Stroke Window. Thus, with no window selected the only operations of note are creating new windows, destroying old ones, or changing the focus.

- Create motive and stroke windows -- Select *Motive Window Button*, or *Stroke Window Button,* and then, click on a blank region of the canvas.
- Delete an existing window -- Move the mouse so that it hover above the unfocused window, and press the Delete key.
- Transfer focus to an existing window by clicking on it. The program will operate according to sections B.c or B.d, depending on whether you send focus to a Stroke Window or a Motive Window.

B.c When A Motive Window has Focus

- Move or resize the window by clicking-and-dragging on any static region of the window's border. Where the cursor changes to a *Move Icon*, you know you can move the window. On the right edge of the window is a stippled region where the cursor will change to a *Right-Left Resize Icon,* click-and-drag there to resize the window.
- Play the motive by hitting space bar, or clicking the play button.
- Change the color associated with a motive by clicking on the two little arrows next to the Motive Window's color swatch.

B.c.a With the Droplet Selected

Place notes by clicking-and-*holding* on a blank region of the window's work area. Grab an existing note by clicking-and-*holding* on an existing note. While your mouse remains held down, your note remains *grabbed* in a temporary state.

- Change the duration of a grabbed note, by pressing the arrow keys while you have a note grabbed, with the mouse button down.
- Move a grabbed note by moving it around.
- Delete the grabbed note by dragging it off the Motive Window.

While you move the grabbed note, any notes which overlap the notes in time display in a faded style. If you release the grabbed note, faded notes will vanish.

B.c.b With the Arrow Selected

Select a note by clicking on it, or by dragging a rectangle around a group of notes. You can add to the selection set by shift-clicking on a note, or shift-dragging a rectangle around a group of notes. When a group of notes is selected, a red box is displayed around them. Finally, if a group of notes has been *copied* onto the clipboard, you can *paste* these notes into the Motive Window by pressing Ctrl-V; the notes appear selected with a box around them.

- Move all the selected notes as a unit by clicking-and-dragging on one of the selected notes, or by using the arrow keys.
- Copy the selected notes onto the clipboard by pressing Ctrl-C. This permits you to *paste* a duplicate of this set later onto any Motive Window, by pressing Ctrl-V for paste.
- Delete the selected notes by pressing Ctrl-X, for *cut*, or by pressing the Delete key.

B.c.c With the Pen Selected

No operations are possible. It's actually a bug that the Pen Button does not gray out completely while focus is on a Motive Window.

B.c.d The Motive-Window or Stroke-Window-Button

Selecting either of the window creation buttons causes focus to drop immediately. The program will operate according to section B.b.

Clicking on a blank region of the blue-and-white canvas also causes focus to drop, but switches to *Arrow Button* mode. Clicking on another existing window transfers focus to that window.

B.d When A Stroke Window has Focus

- Move or resize the window by clicking-and-dragging on any static region of the window's border. Where the cursor changes to a *Move Icon*, you know you can move the window. On the right edge of the window is a stippled region where the cursor will change to a *Right-Left Resize Icon,* click-and-drag there to resize the window.
- Play the piece by hitting space bar, or clicking the play button.
- Change the harmonization algorithm applied to the notes in the piece, by clicking on the radiobuttons attached to the upper left corner of the window.
- Alter the shape of the *harmonization line* by clicking-and-dragging it.

B.d.a With the Droplet Selected

- Place droplet-shaped chords onto the window by clicking on a blank part of the work area.
- Move chords or strokes by clicking and dragging them.
- Delete chords or strokes by dragging them off the Stroke Window

B.d.b With the Pen Selected

- Draw strokes onto the window by clicking on a blank part of the work area and drawing freely.
- Move chords or strokes by clicking and dragging them.
- *Reshape* strokes using a physical model, by right-clicking and dragging on a stroke.
- Delete chords or strokes by dragging them off the Stroke Window

B.d.c With the Arrow Selected

Select a stroke or chord by clicking on it, or by dragging a rectangle around a group of strokes or chords. You can add to the selection set by shift-clicking on an entity, or shift-dragging a rectangle around a group of entities. When a group of strokes and notes is selected, a red box is displayed around them, with three buttons near the upper right corner.

Finally, if a group of strokes or chords has been *copied* onto the clipboard, you can *paste* these into the Stroke Window by pressing Ctrl-V; the entities appear selected with a box around them.

- Move all the selected objects as a unit by clicking-and-dragging on one of the selected objects, or by using the arrow keys.
- Copy the selected objects onto the clipboard by pressing Ctrl-C. This permits you to *paste* a duplicate of this set later onto any Stroke Window, by pressing Ctrl-V for paste.
- Delete the selected notes by pressing Ctrl-X, for *cut*, or by pressing the Delete key.
- Alter the volume of the selected strokes or chords by clicking-and-dragging-vertically on the volume icon near the selection box.
- Alter the timbre of the selected strokes from bowed to plucked, or vice versa, by clicking on the heart icon near the selection box.
- Toggle the melody flag on selected strokes, by clicking on the "plus sign" icon near the selection box.
- Alter the color of the selected objects by clicking on one of the color buttons.

B.d.d The Motive-Window or Stroke-Window-Button

Selecting either of the window creation buttons causes focus to drop immediately. The program will operate according to section B.b.

Clicking on a blank region of the blue-and-white canvas also causes focus to drop, but switches to *Arrow Button* mode. Clicking on another existing window transfers focus to that window.

B.e Navigation

- Pushing the F1--F4 buttons causes the workspace to zoom to one of four zoom levels.
- Holding down the CTRL key allows the user to Click-and-Drag to pan the entire canvas.

B.f Miscellaneous Commands

- The X button, at the lower left corner of the screen, erases all items on the canvas.

- Clicking on a safe saves the current workspace to the hard disk, and loads the workspace corresponding to the clicked safe. This allows the user to rapidly switch between five separate workspaces.

- Press the Q key to quit the program.

- Pressing the Ctrl-O key opens a file dialog allowing the user to select a Hyperscore File to load into the workspace. Pressing Ctrl-S opens a file dialog allowing the user to save the current workspace to a new file.

- Pressing the Ctrl-M key opens a file dialog allowing the user to save a MIDI file, containing the playback music corresponding to whichever window has focus.

- The tempo slider can always be adjusted with the mouse, or with the A and Z shortcut keys.

- Pressing the number keys switches the monitor used to display Hyperscore, in the case that Hyperscore is running on a multi-monitor system.

References

Aimi, R.; *New Expressive Percussion Instruments*; M.S. Thesis 2002, MIT Media Laboratory, Cambridge, MA

Bamberger, J; *Developing Musical Intuitions*; Oxford University Press: New York 2000

Clynes, M; *Sentics*; Prism Press: Bridpot, England 1982

Cope, D.; *The Algorithmic Composer*, A-R Editions, Madison, Wisconsin, 1996

Downie, M; *Behaviour, Animation and Music: The Music and Movement of Synthetic Characters*; M.S. Thesis 2001, MIT Media Laboratory, Cambridge, MA

Farbood, Mary; *Hyperscore: A New Approach to Interactive, Computer Generated Music*; M. S. Thesis 2001, MIT Media Laboratory, Cambridge, MA

Finale; http://www.codamusic.com/finale/

Finkelstein, A; and David H. Salesin; *Multiresolution Curves*, SIGGRAPH 1994 Conference Proceedings

Iwai, T.; *Musical Insects*, 1996

Landay, J., *SILK: Sketching Interfaces Like Krazy*, Proceedings of Human Factors in Computing Systems (Conference Companion), ACM CHI '96, Vancouver Canada, April 13-18, 1996, pp. 398-399.

Langer, S. (1942) Philosophy in a New Key, Cambridge, Harvard University Press

Levin, G.; *Painterly interfaces for audiovisual performance*, M. S. Thesis 2000, MIT Media Laboratory, Cambridge, MA

Machover, T. and Ariane Martins. Toy Symphony Summary Document, 2001

Machover, T., *Brain Opera*, 1996

Matthews, M. and L. Rosler; *Graphical Language for the Scores of Computer-Generated Sounds*, Perspective of New Music, 6(2):92-118, 1968

New York Times, *From a Few Colored Lines Come the Sounds of Music;* May 27th, 2002

Orth, M; *Sculpted Computational Objects, with Smart and Active Computing Materials*, Ph.D. Thesis 2001; MIT Media Laboratory, Cambridge, MA

Perlin, K; Bederson, B., Hollan, J., Meyer, J., Bacon, D., and Furnas, G.; *Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics*, Journal of Visual Languages and Computing, 7, 3-31, 1996.

Pook, S., E. Lecolinet, G. Vaysseix, and E. Barillot. Context and interaction in zoomable user interfaces. In Proc. Advanced Visual Interfaces 2000 (AVI'2000), pages 227--231, 2000

Press, W; and S. Teukolsky; W. Vetterling, B. Flannery, ed.; *Numerical Recipes in C: The Art of Scientific Computing*; 2nd Edition, 1992

Rashmi, T.; *Use of Zooming in Small Screen Interfaces*; M.S. Thesis 2001; University of Toledo

Rice, P. W.; *Stretchable Music: A Graphically Rich, Interactive Composition System*, M.S. Thesis 1998, MIT Media Laboratory, Cambridge, MA

Sibelius, Music scoring software, available at http://www.sibelius.com

Subotnick, M. *Creating Music*, http://www.creatingmusic.com, 1999.

T. Igarashi, and S. Matsuoka, H. Tanaka. *Teddy: A Sketching Interface for 3D Freeform Design*, SIGGRAPH 99 Conference Proceedings

Toy Symphony, http://www.toysymphony.net

Wacom Corporation, *Cintiq*; http://www.wacom.com

Weinberg, G.; *Expressive Digital Musical Instruments For Children,*. M.S. Thesis 1999, MIT Media Laboratory, Cambridge, MA

Wenger, Eric; *MetaSynth*; http://www.metaynth.com

Zeleznik R.C. and K. P. Herndon, J. F. Hughes; *SKETCH: An Interface for Sketching 3D Scenes*, SIGGRAPH 1996 Conference Proceedings